UNIVERSIDADE FEDERAL DE CATALÃO (UFCAT)
INSTITUTO DE MATEMÁTICA E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM MODELAGEM E OTIMIZAÇÃO

ALEXSANDER LINDOLFO DE LIMA

CLASSIFICAÇÃO DE IMAGENS UTILIZANDO REDES DE KOLMOGOROV-ARNOLD PARA DIAGNÓSTICO DE DOENÇAS PULMONARES

CATALÃO (GO)
2025

UNIVERSIDADE FEDERAL DE CATALÃO

INSTITUTO DE MATEMÁTICA E TECNOLOGIA

Av. Dr. Lamartine Pinto de Avelar, número 1120, - Bairro Setor Universitário, Catalão/GO, CEP 75704-020
Telefone: - - https://www.ufcat.edu.br

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO (TECA)

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO (TECA) PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TESES E DISSERTAÇÕES NA BIBLIOTECA DIGITAL DE TESES E DISSERTAÇÕES DA UNIVERSIDADE FEDERAL DE CATALÃO (UFCAT)

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Catalão (UFCAT) a disponibilizar, gratuitamente, por meio da Biblioteca Digital de Teses e Dissertações (BDTD/UFCAT), sem ressarcimento dos direitos autorais, de acordo com a Lei 9.610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou download, a título de divulgação da produção científica brasileira, a partir desta data.

O conteúdo das Teses e Dissertações disponibilizado na BDTD/UFCAT é de responsabilidade exclusiva do autor. Ao encaminhar o produto final, o(a) autor(a) e o(a) orientador(a) Ao encaminhar o produto final, o autor(a) e o(a) orientador(a) firmam o compromisso de que o trabalho não contém nenhuma violação de quaisquer direitos autorais ou outro direito de terceiros.

1. Identificação do material bibliográfico

Dissertação ou Tese? Dissertação

2. Nome completo do autor: ALEXSANDER LINDOLFO DE LIMA

Nome completo do(a) orientador(a): José dos Reis Vieira de Moura Junior

3. Título do trabalho

Título: CLASSIFICAÇÃO DE IMAGENS UTILIZANDO REDES DE KOLMOGOROV-ARNOLD PARA DIAGNÓSTICO DE DOENÇAS PULMONARES

4. Informações de acesso ao documento (este campo deve ser preenchido pelo orientador)

Concorda com a liberação total do documento: [X] SIM [ ] NÃO[1]

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. Após esse período, a possível disponibilização ocorrerá apenas mediante:

a) consulta ao(à) autor(a) e ao(à) orientador(a);

b) novo Termo de Ciência e de Autorização (TECA) assinado e inserido no arquivo da tese ou dissertação.

O documento não será disponibilizado durante o período de embargo.

Casos de embargo:

- Solicitação de registro de patente;

- Submissão de artigo em revista científica;

- Publicação como capítulo de livro;

- Publicação da dissertação/tese em livro.

Obs.: Este termo deverá ser assinado no SEI pelo orientador e pelo autor

Documento assinado eletronicamente por **JOSE DOS REIS VIEIRA DE MOURA JUNIOR**, **Usuário Externo**, em 22/04/2025, às 13:47, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015 .

Documento assinado eletronicamente por **Alexsander Lindolfo de Lima**, **Usuário Externo**, em 22/04/2025, às 23:13, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://sei.ufcat.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **0187273** e o código CRC **B3AE49A1**.

Referência: Processo nº 23852.002421/2025-14 SEI nº 0187273

ALEXSANDER LINDOLFO DE LIMA

CLASSIFICAÇÃO DE IMAGENS UTILIZANDO REDES DE KOLMOGOROV-ARNOLD PARA DIAGNÓSTICO DE DOENÇAS PULMONARES

Dissertação apresentada ao Programa de Pós-Graduação em Modelagem e Otimização, do Instituto de Matemática e Tecnologia, da Universidade Federal de Catalão (UFCAT), como requisito para obtenção do título de Mestre em Modelagem e Otimização. Área de concentração: Modelagem e Otimização.

Orientadora: Prof. Dr. José dos Reis Vieira de Moura Junior.

CATALÃO (GO)
2025

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFCAT.

Lima, Alexsander Lindolfo de
Classificação de imagens utilizando Redes de Kolmogorov-Arnold para diagnóstico de doenças pulmonares / Alexsander Lindolfo de Lima. - 2025.
100, C f.

Orientador: Prof. Dr. José dos Reis Vieira de Moura Junior.
Dissertação (Mestrado) - Universidade Federal de Catalão, Instituto de Matemática e Tecnologia, Catalão, Programa de Pós-Graduação em Modelagem e Otimização, Catalão, 2025.

1. Classificação de imagens. 2. Radiografia do tórax. 3. Doenças pulmonares. 4. Rede de Kolmogorov-Arnold. I. Moura Junior, José dos Reis Vieira de, orient. II. Título.

CDU 5

UNIVERSIDADE FEDERAL DE CATALÃO
Av. Dr. Lamartine Pinto de Avelar, número 1120, - Bairro Setor Universitário, Catalão/GO, CEP 75704-020
Telefone: - - https://www.ufcat.edu.br

ATA DE DEFESA DE DISSERTAÇÃO

Ata nº 01/2025 da sessão de Defesa de Dissertação de **Alexsander Lindolfo de Lima**, que confere o título de **Mestre(a) em Modelagem e Otimização**, na área de concentração em **Modelagem e Otimização.**

Aos dezessete dias de abril de 2025, a partir das **08h00**, Sala //teams.microsoft.com/l/meetup-join, reuniram-se os componentes da banca examinadora, professores **Dr. José dos Reis Vieira de Moura Junior (PPGMO / IMTec / UFCAT), (orientador), Dra. Nádia Félix Felipe da Silva (PPGMO / IMTec / UFCAT), membro titular interno e Dr. José Waldemar da Silva (IME/UFU) , membro titular externo,** para, em sessão pública, procederem a avaliação da Dissertação intitulada "CLASSIFICAÇÃO DE IMAGENS UTILIZANDO REDES DE KOLMOGOROV-ARNOLD PARA DIAGNÓSTICO DE DOENÇAS PULMONARES", de autoria de **Alexsander Lindolfo de Lima**, discente do Programa de Pós-graduação em Modelagem e Otimização – PPGMO, da Universidade Federal de Catalão - UFCAT. A sessão foi aberta pelo presidente, que fez a apresentação formal dos membros da banca. Em seguida, a palavra foi concedida ao discente que, em 42 (quarenta e dois) minutos, procedeu a apresentação. Terminada a apresentação, cada membro da banca arguiu o examinando. Terminada a fase de arguição, procedeu-se a avaliação da Dissertação, que foi considerado: **(X) Aprovado(a) ou ( ) Reprovado(a)**. Cumpridas as formalidades de pauta, a presidência da mesa encerrou a sessão e, para constar, lavrou-se a presente ata que, depois de lida e aprovada, segue assinada pelos membros da banca examinadora.

Documento assinado eletronicamente por **JOSE DOS REIS VIEIRA DE MOURA JUNIOR**, **Usuário Externo**, em 17/04/2025, às 11:05, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015 .

Documento assinado eletronicamente por **José Waldemar da Silva**, **Usuário Externo**, em 17/04/2025, às 11:13, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Nádia Félix Felipe da Silva**, **Usuário Externo**, em 17/04/2025, às 17:31, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://sei.ufcat.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **0186514** e o código CRC **46DBECE5**.

Referência: Processo nº 23852.002421/2025-14 SEI nº 0186514

RESUMO

Este trabalho tem como objetivo desenvolver um sistema de classificação de imagens obtidas a partir da radiografia do tórax para o diagnóstico das doenças pulmonares Pneumonia e Tuberculose, utilizando algoritmos baseados na Rede de Kolmogorov-Arnold (*Efficient* e *Fast*). A pesquisa inicia com uma revisão teórica das doenças pulmonares e do sistema respiratório, estabelecendo a fundamentação necessária para a aplicação de técnicas de aprendizado de máquina. A metodologia envolve o pré-processamento das imagens por meio de técnicas de aumento de dados, a construção de modelos personalizados baseados em CNN, Efficient KAN e Fast KAN. O desempenho dos modelos é avaliado por métricas de avaliação, permitindo a comparação dos resultados obtidos. Estes resultados indicam que as arquiteturas baseadas na Rede de Kolmogorov-Arnold se demonstraram promissoras em comparação com os testes realizados para o modelo CNN, onde foi evidenciado que a técnica *Fast* KAN, com um tempo menor, igualou as porcentagens das métricas de avaliação com o algoritmo CNN, com diferença de 12 horas.

Palavras-chaves: Classificação de imagens, Radiografia do tórax, Doenças pulmonares, Rede de Kolmogorov-Arnold.

ABSTRACT

This work aims to develop a classification system for images obtained from chest x-ray for the diagnosis of lung diseases Pneumonia and Tuberculosis, using algorithms based on the Kolmogorov-Arnold Network (Efficient and Fast). The research begins with a theoretical review of lung and respiratory system diseases, establishing the necessary foundation for the application of machine learning techniques. The methodology involves the pre-processing of images through data augmentation techniques, the construction of customized models based on CNN, Efficient KAN and Fast KAN. The performance of the models is evaluated by evaluation metrics, allowing the comparison of the results obtained. These results indicate that the architectures based on the Kolmogorov-Arnold Network proved to be promising compared to the tests carried out for the CNN model, where it was shown that the Fast KAN technique, with a shorter time, equaled the percentages of the evaluation metrics with the CNN algorithm, with a difference of 12 hours.

Keywords: Image classification, Chest X-ray, Lung diseases, Kolmogorov–Arnold Network.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

AdaGrad — *Adaptive Gradient Algorithm.* Em português: Algoritmo de Gradiente Adaptativo

ADAM — *Adaptive Moment Estimation.* Em português: Estimativa de Momento Adaptativo

AVC — Acidente Vascular Cerebral

BCE — *Binary Cross-Entropy.* Em português: Entropia Cruzada Binária

CCE — *Categorical Cross-Entropy.* Em português: Entropia Cruzada Categórica

CNN — *Convolutional Neural Network.* Em português: Redes Neurais Convolucionais

CRNN — *Convolutional Recurrent Neural Network*

DenseNet-121 — *Densely Connected Convolutional Network with 121 layers.* Em português: Rede Convolucional Densamente Conectada contendo 121 camadas

DenseNet-201 — *Densely Connected Convolutional Network with 201 layers.* Em português: Rede Convolucional Densamente Conectada contendo 201 camadas

FN — *False Negatives*

FP — *False Positives*

HIV — *Human Immunodeficiency Virus.* Em português: Vírus da Imunodeficiência Humana

InceptionV3 — *Inception Version 3*

KAN — *Kolmogorov-Arnold Network.* Em português: Redes de Kolmogorov-Arnold

KNN — *K-Nearest Neighbors.* Em português: K-Vizinhos mais Próximos

LFA — *Line Feature Analysis*

MSE — *Mean Squared Error.* Em português: Erro Quadrático Médio

OMS — Organização Mundial da Saúde

ORB — *Oriented FAST and Rotated BRIEF*

PDE-KAN — *Partial Differential Equation-KAN*

ReLU — *Rectified Linear Unit.* Em português: Unidade Linear Retificada

ResNet-50 — *Residual Network with 50 layers.* Em português: Redes Residuais contendo 50 camadas

ResNet-SVM — Residual Network-Support Vector Machine. Em português: Redes Residuais-Máquina de Vetores de Suporte

RMSProp — *Root Mean Square Propagation.* Em português: Propagação da Raiz Quadrada Média

RNA — Redes Neurais Artificiais

RNN — *Recurrent Neural Networks*

SGD — *Stochastic Gradient Descent.* Em português: Descida do Gradiente Estocástico

SURF — *Speeded-Up Robust Features*

T-KAN — *Temporal-KAN*

TC — Tomografia computadorizada

TP — *True Positives*

US — Ultrassonografia

VGG-16 — *Visual Geometry Group with 16 layers*

VGG-19 — *Visual Geometry Group with 19 layers*

Xception — *Extreme Inception*

LISTA DE CÓDIGOS

SUMÁRIO

Capítulo 1

INTRODUÇÃO

Segundo a Organização Mundial da Saúde (OMS) (2023), no ano de 2021, das dez principais causas de mortes no mundo, cinco são doenças respiratórias, sendo elas: COVID-19 (2º Lugar); doença pulmonar obstrutiva crônica (DPOC) (4º Lugar); infecções agudas das vias respiratórias inferiores, como, por exemplo, pneumonia, bronquite e bronquiolite (5º Lugar); Câncer (Traqueia, Brônquio e Pulmão) (6º Lugar) e Tuberculose (10º Lugar). As outras causas são: Doença cardíaca isquêmica (1º Lugar); acidente vascular cerebral (AVC) (3º Lugar); Doença de Alzheimer e outras demências (7º Lugar); diabete (8º Lugar) e Doença renal (9º Lugar).

Doenças pulmonares são um conjunto de distúrbios que comprometem a integridade do pulmão, sendo este, um grande desafio na área da saúde. Dentre as diversas enfermidades que afetam este órgão, tuberculose e pneumonia estão entre as dez principais causas de mortes no mundo, onde o principal alvo são pessoas vulneráveis, como idosos e pessoas com sistema imunológico fraco (SAÚDE, 2023; PATEL *et al.*, 2024).

A pneumonia é uma infecção pulmonar causada principalmente por bactérias e vírus que provocam inflamação nos alvéolos pulmonares, ou seja, inflamação nos sacos de ar dos pulmões. Esta infecção pode gerar sintomas como tosse, febre, dificuldade para respirar e dor no peito. Para auxiliar os médicos no processo de diagnóstico, exames de imagens obtidas a partir da radiografia do tórax são frequentemente utilizadas devido a sua eficácia em detecção de inflamações pulmonares, enquanto a tomografia computadorizada é utilizada para diagnósticos de problemas mais graves a partir de imagens mais detalhadas utilizando processamento por computador e maiores níveis de radiação (HAQUE *et al.*, 2024; ANTHIMOPOULOS *et al.*, 2016).

A tuberculose é uma infecção causada pela bactéria *Mycobacterium tuberculosis* que, na sua condição inicial, afeta as células alveolares macrofágicas, que por fim, acarretam lesões patológicas decorrentes da aglutinação de células imunes inatas e adaptativas (LAI *et al.*, 2024). Para o diagnóstico desta doença, imagens obtidas a partir da radiografia do tórax

são utilizadas, mas devido à complexidade de identificar esta infecção em relação a outras doenças pulmonares, profissionais treinados e vários testes são necessários para garantir um tratamento eficaz (SATHITRATANACHEEWIN; SUNANTA; PONGPIRUL, 2020; RAHMAN *et al.*, 2020).

Diante das complexidades de se interpretar conjuntos de imagens a partir da radiografia do tórax, que são regularmente utilizadas em tratamentos médicos, profissionais sem experiência podem enfrentar dificuldades em realizar uma avaliação mais precisa. Por outro lado, médicos com experiência também podem ter dificuldade em distinguir a doença pulmonar, o que exige técnicas mais apuradas para auxiliar na avaliação e acompanhamento de pacientes, dentre estas, algoritmos de aprendizado de máquina baseado em Redes Neurais Artificiais (RNA) (FERNÁNDEZ-MIRANDA *et al.*, 2024; HASAN *et al.*, 2024; KIEU *et al.*, 2018).

RNAs se inspiram no funcionamento do cérebro humano para processar dados e aprender com exemplos. Este aprendizado ocorre pelo algoritmo de retro-propagação, que calcula o gradiente da função de custo, permitindo que os pesos sejam ajustados para minimizar a perda durante a etapa de treinamento. Neste contexto, as Redes Neurais Convolucionais (CNN - *Convolutional Neural Network*) têm sido amplamente utilizadas nas áreas de processamento de imagens e visão computacional (LECUN; BENGIO; HINTON, 2015; JIAO; ZHAO, 2019). Por meio das camadas convolucionais, que funcionam como filtros, é possível analisar estruturas de dados bidimensionais para gerar novos mapas de características (LECUN *et al.*, 1998; JIAO; ZHAO, 2019).

Rede de Kolmogorov-Arnold (KAN - *Kolmogorov-Arnold Network*) foi criada pelos autores Liu et al. (2024b) e descrita como uma alternativa promissora em comparação as RNAs tradicionais, que têm funções de ativação fixas nos neurônios artificiais com o valor dos pesos das conexões sendo linear. Em contrapartida, nesta nova abordagem, às funções de ativação são ajustadas nas conexões, sendo os neurônios artificiais responsáveis pelas operações de somatório.

Considerando o mencionado, com este trabalho, pretende-se desenvolver um sistema de classificação de imagens obtidas a partir da radiografia do tórax para diagnóstico de doenças pulmonares utilizando KAN. De acordo com Hou e Zhang (2024), estas redes podem capturar correlações complexas para problemas de classificação de imagens médicas devido à sua capacidade de modelar funções complexas. Para o desenvolvimento do problema proposto, a lista a seguir apresenta as etapas a serem concluídas.

- Realizar o pré-processamento das imagens utilizando técnicas de aumento de dados para simular condições reais que podem acontecer durante a captura das imagens;

- Desenvolver um classificador para as doenças pulmonares Pneumonia e Tuberculose utilizando um modelo CNN personalizado;

- Desenvolver um classificador para as doenças pulmonares Pneumonia e Tuberculose utilizando os algorítimos *Efficient* KAN e *Fast* KAN, baseados na arquitetura KAN;
- Avaliar o desempenho dos modelos no diagnóstico de pneumonia e tuberculose;
- Comparar os resultados apresentados pelos classificadores desenvolvidos;
- Elaborar a parte escrita da dissertação bem como a apresentação de defesa.

1.1 Estrutura da dissertação

O Capítulo 2, apresenta uma ampla variedade de artigos em que foram utilizadas diversas técnicas de aprendizado de máquina para identificação de doenças pulmonares. Por fim, diversos outros artigos foram apresentados onde o algoritmo KAN foi utilizado como base para desenvolver técnicas para problemas específicos.

O Capítulo 3, foi dividido entre as seções Sistema respiratório, Doenças pulmonares, Diagnóstico por imagens, Redes neurais artificiais, Redes de Kolmogorov-Arnold, *Efficient* KAN, *Fast* KAN, Pré-processamento de imagens e Métricas de avaliação. Nestas seções foram descritas as fundamentações teóricas sobre os temas que abrangem este trabalho e que são necessárias para alcançar o objetivo.

O Capítulo 4, apresenta a maneira em que foi construído o banco de dados, o equipamento utilizado bem como a linguagem de programação e as bibliotecas, a implementação do pré-processamento das imagens, as implementações dos modelos nas três etapas de treinamento-validação e a implementação das métricas de avaliação.

O Capítulo 5, apresenta os resultados das implementações descritas no Capítulo 4 para cada modelo nas três etapas. Estas diferentes situações de implementações foram necessárias para desenvolver uma análise empírica de acordo com dados reais e observações diversificadas.

O Capítulo 6, apresenta as conclusões em relação a uma análise comparativa entre os resultados apresentados no Capítulo 5, bem como trabalhos futuros que podem ser explorados em relação às evidências encontradas neste trabalho.

Capítulo 2

REFERENCIAL TEÓRICO

Devido ao grande número de artigos publicados com foco em classificação de doenças pulmonares utilizando técnicas de aprendizado de máquina, neste capítulo é apresentado alguns artigos em que os autores utilizaram tanto modelos personalizados de CNN quanto modelos obtidos a partir de aprendizado por transferência, modelo pré-treinado que pode ser reutilizado (PRATT, 1992). Também é apresentado alguns estudos que envolvem o uso do algoritmo KAN para problemas específicos.

Arias-Londoño (2020), utilizaram CNN para detecção automática de COVID-19 por meio de imagens de radiografia do tórax. O objetivo deste estudo foi avaliar como técnicas de pré-processamento de imagens podem afetar os resultados e comprometer o sistema, simulando diferentes tipos de situações em que as imagens podem estar. Como resultado, foi possível alcançar uma precisão de 91,5% e 87,4% de *average recall*.

Ausawalaithong et al. (2018), utilizaram a Rede Convolucional Densamente Conectada contendo 121 camadas (DenseNet-121 - *Densely Connected Convolutional Network with 121 layers*) para identificar câncer no pulmão em imagens de radiografia do tórax de um banco de dados com poucas amostras. O algoritmo proposto alcançou, em média, uma acurácia de mais de 70%, demonstrando a eficácia desta abordagem em relação ao diagnóstico de câncer no pulmão.

Muñoz-Aseguinolaza et al. (2023), utilizaram modelos de CNN para análise de imagens médicas em 3D, do câncer de pulmão. Superando as limitações dos métodos de classificação de imagens 2D, nesta abordagem foi possível extrair informações relevantes de dados tridimensionais, com a melhor performance alcançando 76,44% de acurácia na classificação do câncer de pulmão.

Hamal et al. (2024), utilizaram os modelos, K-Vizinhos mais Próximos (KNN - *K-Nearest Neighbors*), Árvores de Decisão, CNN, *Visual Geometry Group with 16 layers* (VGG-16), *Visual Geometry Group with 19 layers* (VGG-19), Rede Convolucional Densamente Conectada contendo 201 camadas (DenseNet-201 - *Densely Connected Convolutional Network*

with 201 layers), Redes Residuais contendo 50 camadas (ResNet-50 - *Residual Network*), *Inception Version 3* (InceptionV3) e *Extreme Inception* (Xception) para detecção da COVID-19 a partir de imagens de raio-X. Foi demonstrado que o modelo VGG-19 obteve uma precisão de 99% no treinamento e 98% nos testes, chegando em 90% na detecção de COVID-19, 90% na detecção do estado normal do pulmão e 100% da detecção de pneumonia, sendo este o modelo que alcançou o melhor desempenho entre as técnicas testadas, comprovando a eficácia desta abordagem na classificação de doenças pulmonares em imagens de raio-X.

Malik et al. (2023), propuseram um modelo que combina técnicas para extração de características ORB (*Oriented FAST and Rotated BRIEF*) e SURF (*Speeded-Up Robust Features*) com um modelo CNN (VGG-19) para classificação de doenças pulmonares utilizando imagens de radiografia do tórax. O modelo proposto alcançou uma precisão de 98,20% em relação à classificação das doenças pulmonares COVID-19, câncer no pulmão, Atelectasia, Consolidação pulmonar, Tuberculose, Pneumotórax, Edema, Pneumonia e Espessamento pleural.

Kim, Hong e Park (2021), propuseram um modelo de detecção de COVID-19 em imagens de radiografia de tórax que utiliza técnicas de redução de dimensionalidade e detecção de bordas. Este modelo consiste na combinação do algoritmo Análise de Características de Linha (LFA - *Line Feature Analysis*), que faz a extração de informações sobre a aparência das linhas, com a técnica Rede Neural Recorrente (RNN - *Recurrent Neural Networks*), aprendizado utilizando dados sequenciais. Para validar o modelo proposto, comparações com outros modelos foram realizadas pelos autores, sendo elas: Rede Neural Convolucional Recorrente (CRNN - *Convolutional Recurrent Neural Network*), VGG, AlexNet, Conv1D e DNN. Utilizando a métrica Acurácia, o modelo LFA-RNN alcançou 97,5/%, sendo este o melhor resultado. Para os outros modelos, os resultados foram: 96,1% para CRNN, 96,6% para VGG, 94,1% para AlexNet, 79,4% para Conv1D e 78,9% para DNN.

Alshmrani et al. (2023), utilizaram a arquitetura de aprendizado profundo, VGG-19 e CNN, para classificação de pneumonia, câncer de pulmão, tuberculose, opacidade pulmonar e COVID-19 por meio de imagens de raio-x. Após feito o pré-processamento das imagens requeridas pela arquitetura, a aplicação da metodologia proposta alcançou um desempenho de 96,48% de acurácia, 93,75% de *recall*, 97,56% de precisão, 95,62% de *F1-score* e 99,82% de ROC-AUC, demonstrando que este modelo alcançou um grande desempenho, fazendo com que diagnósticos de doenças pulmonares sejam feitos de maneira rápida e precisa.

Zhou et al. (2021), utilizaram reagrupamento de imagens e Redes Residuais com Máquina de Vetores de Suporte (ResNet-SVM - *Residual Network-Support Vector Machine*) para detecção de COVID-19 a partir de imagens da radiografia do tórax. A aplicação da metodologia proposta resultou em um desempenho de 93% de precisão, com poucas imagens no banco de dados referente ao treinamento.

Devido a recente publicação feita pelos autores Liu et al. (2024b) sobre o algoritmo

KAN, artigos propondo a utilização desta técnica voltados ao campo da saúde ainda não foram divulgados. Por outro lado, em outra publicação feita pelos autores Liu et al. (2024a), é mencionado que diversas pesquisas foram realizadas visando desenvolver aplicações do algoritmo KAN em outros campos, em que as considerações finais mencionadas pelos autores ressaltam a eficácia desta técnica em comparação às técnicas de RNAs tradicionais, como, por exemplo, aprendizado por grafos (KIAMARI; KIAMARI; KRISHNAMACHARI, 2024; BRESSON *et al.*, 2024; CARLO; MASTROPIETRO; ANAGNOSTOPOULOS, 2024), visão computacional (CHEON, 2024; FIRSOV *et al.*, 2024), séries temporais (VACA-RUBIO *et al.*, 2024) e utilizando outras funções unidimensionais como, por exemplo, *Wavelet* (BOZORGASL; CHEN, 2024), *Fourier* (XU *et al.*, 2024) e *Jacobi* (AGHAEI, 2025).

De acordo com uma revisão sistemática sobre os fundamentos e aplicações da arquitetura KAN feita pelos autores Somvanshi et al. (2024), as técnicas destacadas como principais avanços em cenários dinâmicos foram *Temporal-KAN* (T-KAN) (XU; CHEN; WANG, 2024), *FastKAN* (LI, 2024) e *Partial Differential Equation*-KAN (PDE-KAN), onde estes modelos apresentaram melhorias na eficiência computacional e na adaptação em relação à resolução de problemas complexos. Por outro lado, devido a sua versatilidade em complementação para cenários que exigem arquiteturas híbridas, o modelo KAN pode ser integrado com estruturas convolucionais (ELAZIZ; FARES; ASEERI, 2024; BODNER *et al.*, 2024; IGALI; SHAMOI, 2024; DROKIN, 2024), recorrentes (GENET; INZIRILLO, 2024) e aqueles baseados em transformadores (XIE *et al.*, 2024).

Capítulo 3

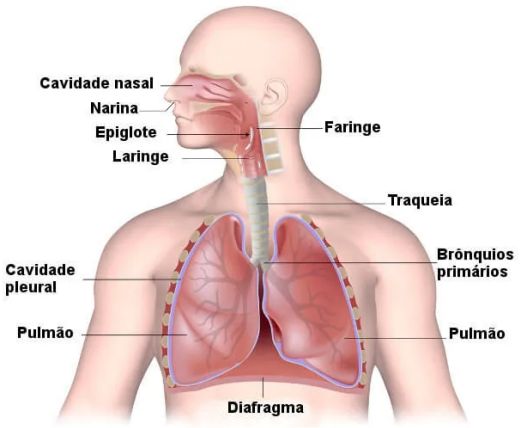
FUNDAMENTAÇÃO TEÓRICA

Neste capítulo é apresentado a fundamentação teórica que servirá como base para este trabalho. Nas Seções 3.1, 3.2 e 3.3, serão levantados os conceitos sobre o sistema respiratório, doenças pulmonares e técnicas de diagnóstico por imagens. Nas Seções 3.4, 3.5, 3.8 e 3.9, serão apresentadas as bases teóricas e métodos de implementação sobre RNA, CNN e KAN, bem como as técnicas de pré-processamento de imagens e métricas de avaliação.

3.1 Sistema respiratório

O sistema respiratório tem como principal função captar o oxigênio e eliminar o gás carbônico, bem como na filtragem e aquecimento do ar. De maneira funcional, este sistema é dividido pelas zonas de condução e respiratória. Por outro lado, de maneira anatômica, a divisão ocorre pelas vias aéreas superiores, localizada fora do tórax e inferiores, localizada dentro do tórax (PATWA; SHAH, 2015). A Figura 3.1 apresenta uma esquematização do sistema respiratório.

Figura 3.1 – Esquematização do sistema respiratório.



Fonte: Retirado de Santos (2024).

3.1.1 Zona de condução e respiratória

Na zona de condução, como o próprio nome diz, tem como finalidade ser a via condutora de entrada e saída de ar até os pulmões, passando por um processo de filtragem, umidificação e aquecimento. A lista a seguir apresenta a sequência estrutural composta por esta zona (PATWA; SHAH, 2015).

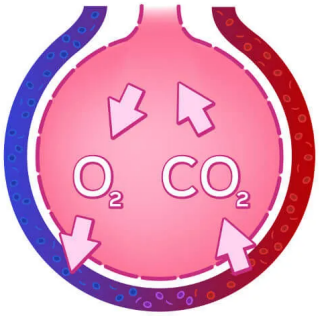
- Nariz,
- Faringe,
- Laringe,
- Traqueia,
- Brônquios,
- Bronquíolos.

Por outro lado, a zona respiratória é onde ocorrem as trocas gasosas. A lista a seguir apresenta a sequência estrutural composta por esta zona (PATWA; SHAH, 2015).

- Bronquíolos respiratórios,
- Ductos alveolares,
- Alvéolos pulmonares.

A troca gasosa ocorre nos alvéolos pulmonares, onde no processo de inspiração, o oxigênio se dissolve na camada de revestimento dos alvéolos, que por sua vez é levado para o sangue, e no processo de expiração, o gás carbônico resultante é expelido pelo sistema respiratório. A Figura 3.2 ilustra as trocas gasosas nos alvéolos pulmonares (PATWA; SHAH, 2015).

Figura 3.2 – Trocas gasosas nos alvéolos pulmonares.



Fonte: Retirado de Santos (2024).

3.2 Doenças pulmonares

Existem diversas doenças pulmonares que afetam os alvéolos pulmonares, responsáveis pela oxigenação do sangue. Para este trabalho, serão consideradas as doenças Pneumonia e Tuberculose, visto que estas doenças aparecem em publicação feita pela Organização Mundial da Saúde (2023) das dez principais causas de mortes do mundo em 2021. A lista a seguir apresenta este *ranking*.

1. Doença cardíaca isquêmica;
2. COVID-19;
3. AVC;
4. DPOC;
5. Infecções respiratórias inferiores (Pneumonia, Bronquite e Bronquiolite);
6. Câncer de Traqueia, Brônquio e Pulmão;
7. Doença de Alzheimer e outras demências;
8. Diabetes;
9. Doença renal;
10. Tuberculose.

Ainda conforme os dados publicados pela Organização Mundial da Saúde (2023), a Tabela 3.1 apresenta as posições das doenças, infecções respiratórias inferiores (Pneumonia, Bronquite e Bronquiolite) e tuberculose, pelo agrupamento de países em relação às condições econômicas.

Tabela 3.1 – Posição das doenças em relação às condições econômicas dos países.

Condições econômicas	**Infecções respiratórias inferiores**	**Tuberculose**
Países de renda baixa	1º	8º
Países de renda média-baixa	5º	6º
Países de renda média-alta	7º	NaN
Países de renda alta	8º	NaN

Fonte: Retirado de Organização Mundial da Saúde (2023).

3.2.1 Pneumonia

A pneumonia faz parte das infecções respiratórias inferiores, causada principalmente por bactérias ou vírus, geralmente transmitidas pela aproximação com pessoas infectadas. As principais vítimas são crianças menores de 5 anos, adultos maiores de 65 anos e pessoas com problemas de saúde preexistentes. Esta infecção faz com que os alvéolos pulmonares fiquem cheios de fluidos, levando a uma absorção de oxigênio limitada. A Figura 3.3 apresenta uma ilustração de um alvéolo pulmonar com presença de fluidos (SAÚDE, 2024a).

Figura 3.3 – Ilustração do alvéolo com fluido.

Fonte: Retirado de Vieira (2021).

A lista a seguir apresenta os sintomas da pneumonia dependendo do estado em que se encontra o indivíduo (SAÚDE, 2024a).

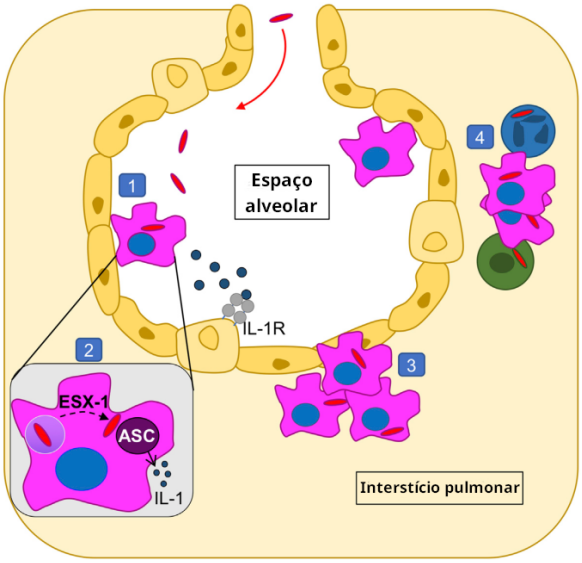
- Tosse;
- Falta de ar;
- Febre;
- Dor no peito;
- Náuseas;
- Confusão mental (Ocorre principalmente em idosos).

O tratamento da pneumonia bacteriana é feito com o uso de antibióticos, como a amoxicilina. A internação hospitalar é necessária apenas quando sintomas mais graves são apresentados, como febre alta, dificuldades respiratórias e comprometimento da função dos rins e da pressão arterial (SAÚDE, 2024a; SAÚDE, 2024). Por outro lado, o tratamento dos sintomas da pneumonia viral é feito utilizando antitérmicos e analgésicos. Em um estado mais grave da pneumonia viral, são utilizados medicamentos como, oseltamivir e remdesivir (EINSTEIN, 2024).

3.2.2 Tuberculose

A Tuberculose afeta os pulmões por meio da bactéria *Mycobacterium tuberculosis*, que na fase inicial, infecta os macrófagos alveolares (Mecanismo de defesa). Esta interação faz com que os macrófagos do espaço alveolar se desloquem para o tecido responsável pelas trocas gasosas (Interstício pulmonar), concretizando a disseminação da doença (COHEN *et al.*, 2018). A Figura 3.4 apresenta estas interações.

Figura 3.4 – Ilustração da interação entre macrófagos e *Mycobacterium tuberculosis*.



Fonte: Adaptado de Cohen et al. (2018).

Como mostrado pela Figura 3.4, a primeira fase representa o primeiro contato entre *Mycobacterium tuberculosis* e os macrófagos alveolares. Na segunda fase, ocorre a secreção (ESX-1) do *Mycobacterium tuberculosis*, sendo este o processo de disseminação desta bactéria. Na terceira fase, estes macrófagos infectados se movem para o interstício pulmonar. Na quarta fase, a infecção se espalha para outras células imunes (COHEN *et al.*, 2018).

As vítimas com maior risco de contrair tuberculose são aquelas com sistema imunológico fraco, como, por exemplo, pessoas portadoras do Vírus da Imunodeficiência Humana (HIV - *Human Immunodeficiency Virus*), pessoas em situação de desnutrição e aqueles que possuem diabete. A lista a seguir apresenta os sintomas da Tuberculose (SAÚDE, 2024b).

- Tosse,
- Dor no peito,
- Fraqueza/fadiga,
- Perda de peso,
- Febre.

O tratamento da tuberculose é feito ao longo de seis meses, onde os principais antibióticos são Rifampicina e Isoniazida. Caso o tratamento utilizando esses medicamentos não apresentem efeitos, devido à resistência da bactéria, tratamentos mais longos e sofisticados são necessários (SAÚDE, 2024b).

3.3 Diagnóstico por imagens

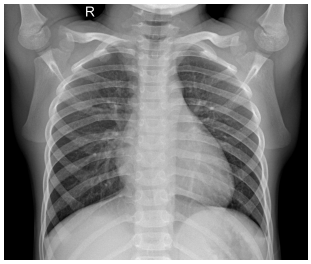
Técnicas de diagnósticos por imagem são regularmente utilizadas para identificar mudanças ocasionadas por doenças e acidentes, que têm como base a utilização de diferentes tipos de energia que atravessaram o corpo humano para captar as condições em que se encontra sua estrutura interna, como, por exemplo, traumas, tumores e alterações estruturais/funcionais. Dentre estas técnicas para o diagnóstico por imagem, as mais utilizadas são apresentadas na lista a seguir (MOORE; DALLEY; AGUR, 2014).

- Radiografia simples (Raio-x);
- Tomografia computadorizada (TC);
- Ultrassonografia (US);

3.3.1 Radiografia simples

O raio-x utiliza feixes contendo baixas doses de radiação que passam através do paciente, mostrando as condições das estruturas corporais em relação às densidades de massa, e de um detector, que receberá o retorno do comportamento destes feixes no corpo. Portanto, órgãos/tecidos que possuem densidades maiores produzem uma área mais brilhante (MOORE; DALLEY; AGUR, 2014). A Figura 3.5 apresenta um exemplo de uma imagem de radiografia do tórax.

Figura 3.5 – Exemplo de uma imagem de radiografia do tórax.



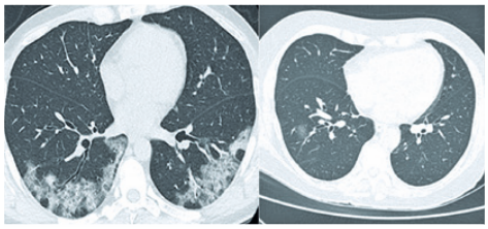
Fonte: Retirado de Kermany et al. (2018).

No contexto deste trabalho, imagens de radiografia do tórax foram utilizadas devido à simplicidade, eficácia, uso recorrente na literatura e por ser mais barata que outras técnicas de diagnóstico por imagens.

3.3.2 Tomografia computadorizada

A produção de imagens pela técnica TC ocorre mediante cortes transversais utilizando feixes de raio-x. A Figura 3.6 apresenta um conjunto de imagens obtidas pela TC do tórax.

Figura 3.6 – Conjunto de imagens da TC do tórax.



Fonte: Adaptado de Fonseca et al. (2021).

Por meio da rotação do tubo de raio-x e do detector, medidas de energia radial são captadas e comparadas utilizando um computador, visando analisar a densidade radiológica da região do corpo em questão. De maneira análoga à radiografia simples, nas imagens da TC, órgãos/tecidos com maior densidade produzem áreas mais brilhantes, enquanto áreas pretas são atribuídas a densidades menores (MOORE; DALLEY; AGUR, 2014).

3.3.3 Ultrassonografia

A produção de imagem pela técnica US ocorre pela geração de ondas sonoras de alta frequência por um transdutor, que atravessam o corpo e refletem as características em tempo real da área em análise, que podem ser exibidas em um monitor por meio de imagens seccionais. A Figura 3.7 apresenta um exemplo de imagem obtida a partir da US obstétrica.

Figura 3.7 – Exemplo de US obstétrica.

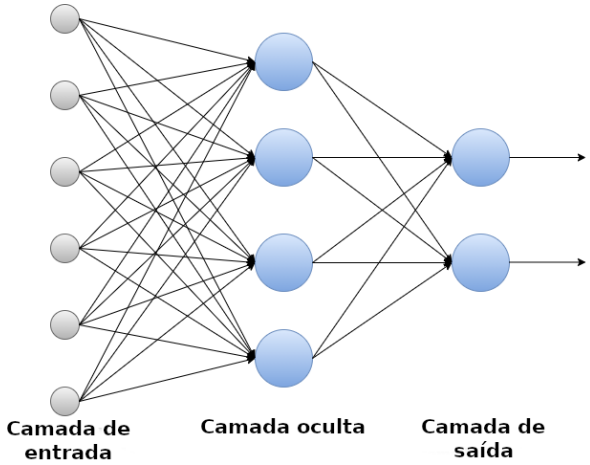
Fonte: Retirado de Diagnósticos (2024).

A condução das ondas de ultrassom não ocorrem de maneira adequada pelo ar, portanto, esta técnica não é utilizada para diagnósticos do pulmão, sendo mais recomendada para exames preventivos durante a gravidez, visto que este procedimento não utiliza radiação (MOORE; DALLEY; AGUR, 2014).

3.4 Redes neurais artificiais

Sistemas do tipo RNA são inspirados no funcionamento do cérebro humano, onde as unidades básicas de processamento (neurônios artificiais) são distribuídas em camadas de entrada (Dados de entrada), oculta (Cálculos intermediários) e saída (Predição final), ilustrado pela Figura 3.8, sendo que a quantidade de camadas e neurônios artificiais bem como suas propriedades variam conforme o problema a ser resolvido (HAYKIN, 2009).

Figura 3.8 – Representação das RNAs.

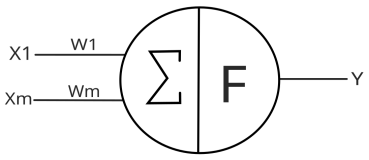


Fonte: Adaptado de Haykin (2009).

3.4.1 Neurônio de McCulloch-Pitts

O conceito de neurônio artificial, em termos de lógica proposicional (AND, OR e NOT), foi introduzido inicialmente por McCulloch e Pitts (1943), onde cada neurônio possui múltiplas entradas e uma saída, sendo somente sinais binários. Os pesos, que determinam a importância da conexão, são fixos e não ajustáveis. Fazendo o somatório dos sinais de entrada ($X1$, ..., Xm) multiplicados pelos seus respectivos pesos ($W1$, ..., Wm), utilizando a função de ativação (F) limiar, caso o valor do somatório ultrapasse o valor definido como limiar, o neurônio será ativado, ou seja, a saída (Y) será igual a 1, caso contrário, será 0. A Figura 3.9 apresenta uma representação gráfica do neurônio de McCulloch-Pitts.

Figura 3.9 – Representação gráfica do neurônio de McCulloch-Pitts.



Fonte: Adaptado de Haykin (2009).

3.4.2 Perceptron de Rosenblatt

Mesmo que o neurônio de McCulloch-Pitts e o perceptron proposto por Rosenblatt (1958) tenham a mesma ideia de simular o funcionamento da estrutura biológica do neurônio, existem diferenças nas suas composições. O Quadro 3.1 apresenta as principais diferenças entre o neurônio de McCulloch-Pitts e o perceptron de Rosenblatt.

McCulloch-Pitts	**Rosenblatt**
Somente entradas binárias	Entradas podem ser números reais
Pesos fixos	Pesos ajustáveis
Resolve problemas em termos de lógica simples	Resolve problemas linearmente separáveis

Fonte: Adaptado de McCulloch-Pitts (1943) e Rosenblatt (1958).

Quadro 3.1 – Diferenças entre neurônio e perceptron.

O perceptron utiliza o mesmo modelo de neurônio artificial ilustrado pela Figura 3.9, mas, como mostrado pelo Quadro 3.1, existe a possibilidade de resolver problemas de classificação binária linearmente separáveis por meio da capacidade de permitir números reais como dados de entrada e pesos ajustáveis por meio do algoritmo de aprendizagem proposto por Rosenblatt, diferente dos neurônios de McCulloch-Pitts, que permitem somente operações lógicas simples, visto que, somente é permitido entradas binárias e pesos fixos que devem ser definidos manualmente (ROSENBLATT, 1958). A equação 3.1 demonstra a modelagem matemática do perceptron.

$$v = \sum_{i=1}^{m} w_i x_i + b, \tag{3.1}$$

onde, v é a representação matemática do perceptron, w_i são os pesos, x_i são os dados de entrada e b é o viés, também chamado de *bias*, responsável por ajustar a posição da fronteira de decisão para melhorar a classificação binária (ROSENBLATT, 1958). O ajuste dos pesos (w e b) ocorre pela correção dos erros mediante a predição dos dados de entrada. A equação 3.2 demonstra a modelagem matemática da atualização dos pesos (ROSENBLATT, 1958).

$$w(i+1) = w(i) + \eta(d(i) - y(i))x(i), \tag{3.2}$$

onde, i é o incremento, $w(i+1)$ é a atualização do peso, $w(i)$ peso atual, η é o valor da taxa de aprendizagem, $(d-y)$ é o cálculo do erro, sendo d a resposta desejada e y a resposta atual, e x os dados de entrada (ROSENBLATT, 1958).

3.4.3 Retro-propagação

Este aprendizado ocorre através do ajuste dos pesos das conexões entre os neurônios artificiais de uma RNA com camadas de entrada, oculta e saída. Neste método, ocorre a minimização de uma função de custo viabilizando a capacidade de generalização, ou seja, dado

um conjunto de dados de entrada, minimizar a diferença entre a saída predita e a saída desejada com base no gradiente do erro, mostrado pela equação 3.3 (RUMELHART; HINTON; WILLIAMS, 1986).

$$E = \frac{1}{2}\sum_{c}\sum_{j}(y_{j,c} - d_{j,c})^2, \tag{3.3}$$

onde, E é o erro, c é o índice sobre os pares de entrada e saída, j é o índice sobre as unidades de saída, y representa o estado real e d é o estado desejado. A minimização do erro, e por consequência, a atualização dos pesos, ocorre mediante ao cálculo de derivadas parciais de E em relação aos pesos, por meio da descida do gradiente. A equação 3.4 demonstra este cálculo (RUMELHART; HINTON; WILLIAMS, 1986).

$$w^* = w - \eta\left(\frac{\partial E}{\partial w}\right), \tag{3.4}$$

onde, w^* é o peso atualizado, w é o peso atual, η é a taxa de aprendizagem (RUMELHART; HINTON; WILLIAMS, 1986).

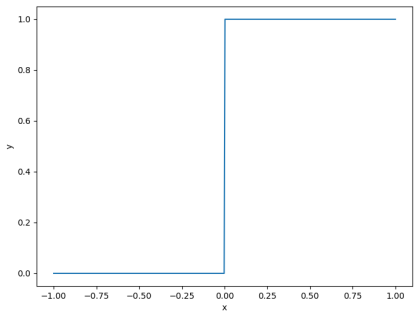
3.4.4 Funções de ativação

As funções de ativação têm como objetivo determinar se a informação contida em um neurônio artificial será passada para a próxima camada da RNA. O sistema 3.5 apresenta a função Limiar, onde θ é o parâmetro ajustável, enquanto o sistema 3.6 apresenta a função Degrau, onde o parâmetro do ponto de corte é fixo, mostrado pela Figura 3.10 (MCCULLOCH; PITTS, 1943; ROSENBLATT, 1958).

$$limiar = \begin{cases} 0\ se\ x < \theta \\ 1\ se\ x \geq \theta \end{cases} \tag{3.5}$$

$$degrau = \begin{cases} 0\ se\ x < 0 \\ 1\ se\ x \geq 0 \end{cases} \tag{3.6}$$

Figura 3.10 – Representação gráfica das funções de ativação Limiar e Degrau.



Fonte: o autor.

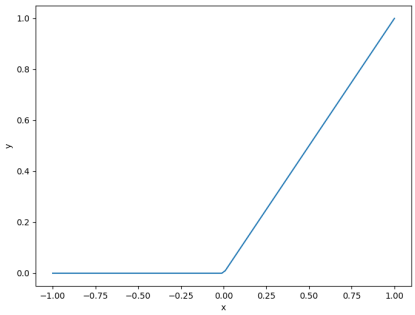
Por outro lado, existem outras funções de ativação que introduzem não-linearidade no modelo, fazendo com que relações mais complexas entre os dados de entrada e saída sejam modeladas para que problemas mais complexos sejam resolvidos, como, por exemplo,

visão computacional. De acordo com Goodfellow, Bengio e Courville (2016), uma das funções de ativação mais utilizada e mais recomendada nos neurônios artificiais das camadas ocultas é a Unidade Linear Retificada (ReLU - *Rectified Linear Unit*) (NAIR; HINTON, 2010), demonstrada pela equação 3.7.

$$ReLU(x) = max(0,\ x), \tag{3.7}$$

onde, x são os dados das camadas anteriores. A Figura 3.11 apresenta o gráfico da função de ativação ReLU.

Figura 3.11 – Gráfico da função ReLU.



Fonte: o autor.

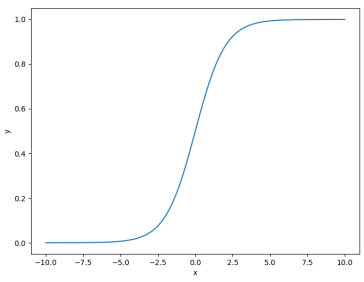
Para os neurônios artificiais da camada de saída, as funções de ativação mais utilizadas são, *Sigmoid*, para classificação binária, apresentado pela equação 3.8 e pela Figura 3.12a e, *Softmax*, para classificação de múltiplas classes, apresentado pela equação 3.9 e pela Figura 3.12b (BISHOP; NASRABADI, 2006).

$$Sigmoid(x) = \frac{1}{1+e^{-x}}, \tag{3.8}$$

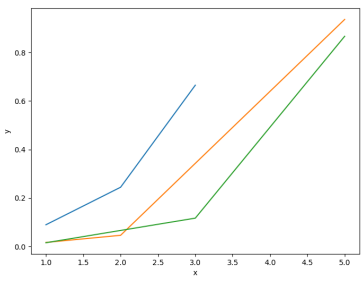
$$Softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}. \tag{3.9}$$

Figura 3.12 – Funções de ativação da camada de saída.

(a) Gráfico da função Sigmoid.



(b) Gráfico da função Softmax.



Fonte: o autor.

3.4.5 Funções de perda

As funções de perda, também chamadas de funções de custo, são utilizadas nas RNAs para quantificar as diferenças entre os resultados encontrados com os observados, onde este valor quantificado será minimizado por meio de técnicas de otimização para ajustar os valores dos pesos (RASCHKA; MIRJALILI, 2017). O Quadro 3.2 apresenta as funções de perda mais utilizadas para resolver problemas de regressão e classificação.

Funções	**Aplicação**
Erro Quadrático Médio (MSE - *Mean Squared Error*)	Regressão
Entropia Cruzada Binária (BCE - *Binary Cross-Entropy*)	Classificação
Entropia Cruzada Categórica (CCE - *Categorical Cross-Entropy*)	Classificação

Fonte: o autor.

Quadro 3.2 – Funções de perda.

A função de perda MSE é utilizada para problemas de regressão, que produz uma saída contínua, ou seja, predição de valores futuros em relação aos dados passados. Esta função calcula a média dos quadrados da diferença entre os resultados encontrados com os verdadeiros (GOODFELLOW; BENGIO; COURVILLE, 2016).

$$MSE = \frac{1}{m}\sum_{i}(d-y)_i^2, \tag{3.10}$$

onde, m é o número total de amostras, d são os resultados encontrados e y são os resultados verdadeiros.

Por outro lado, a função de perda BCE é utilizada para problemas de classificação binária, ou seja, classificação em relação a duas classes distintas. A equação 3.11 apresenta o cálculo utilizando os resultados encontrados e os verdadeiros (BISHOP; NASRABADI, 2006).

$$BCE = -\frac{1}{m}\sum_{i=1}^{m} y_i log(d_i) + (1-y_i) log(1-d_i), \tag{3.11}$$

onde, m é o número total de amostras, d são os resultados encontrados e y são os resultados verdadeiros. Já a função de perda CCE é utilizada para problemas de múltiplas classes, onde a equação 3.12 apresenta o cálculo utilizando os resultados encontrados e os verdadeiros (MURPHY, 2012).

$$CCE = -\sum_{i} y_i log(d_i), \tag{3.12}$$

onde, y são os resultados verdadeiros e d são os resultados encontrados.

3.4.6 Otimizadores

Técnicas de otimização são utilizadas nas RNAs tradicionais para minimizar uma função de perda, ajustando os pesos das conexões entre os neurônios artificiais para aproxi-

mar os resultados encontrados com os verdadeiros (GOODFELLOW; BENGIO; COURVILLE, 2016). O Quadro 3.3 apresenta alguns exemplos de técnicas de otimização com suas respectivas descrições.

Técnicas	Descrições
Descida do Gradiente Estocástico	Parâmetros na direção oposta ao gradiente.
Estimativa de Momento Adaptativo	Médias móveis dos gradientes e seus quadrados.

Fonte: o autor.

Quadro 3.3 – Técnicas de otimização.

Descida do Gradiente Estocástico (SGD - *Stochastic Gradient Descent*) é uma técnica de otimização baseada na técnica Descida do Gradiente, diferindo em relação à atualização dos parâmetros, onde a SGD ocorre após cada amostra, enquanto a técnica Descida do Gradiente ocorre após todo o conjunto de dados. Nesta técnica de otimização, o cálculo do vetor gradiente indica a direção oposta em que deve ocorrer os ajustes dos pesos para reduzir o erro encontrado pelo modelo (LECUN; BENGIO; HINTON, 2015).

Estimativa de Momento Adaptativo (ADAM - *Adaptive Moment Estimation*) é uma técnica que combina outros dois métodos de otimização, sendo eles, Algoritmo de Gradiente Adaptativo (AdaGrad - *Adaptive Gradient Algorithm*) e Propagação da Raiz Quadrada Média (RMSProp - *Root Mean Square Propagation*) (KINGMA, 2014). O Quadro 3.4 apresenta as características adotadas pela técnica ADAM dos métodos AdaGrad e RMSProp.

Técnicas	Funcionamento
AdaGrad	Funciona bem com gradientes esparsos.
RMSProp	Funciona bem em configurações não-estacionárias.

Fonte: o autor.

Quadro 3.4 – Características adotadas por ADAM.

Portanto, o ajuste das taxas de aprendizado adaptativas para diferentes parâmetros ocorre, em um primeiro momento, com base em uma média móvel exponencial dos gradientes, e em um segundo momento, com base na média móvel exponencial dos quadrados do gradiente (KINGMA, 2014). A equação 3.13 demonstra o cálculo do primeiro momento e a equação 3.14 apresenta o cálculo da correção do viés.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla_\theta f_t(\theta_{t-1}), \qquad (3.13)$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \qquad (3.14)$$

onde, t representa a iteração, β_1 é a taxa de decaimento exponencial para estimativas e $f(\theta)$ é a função objetivo em relação aos parâmetros da rede. A equação 3.15 demonstra o cálculo do segundo momento e a equação 3.16 apresenta o cálculo da correção do viés (KINGMA, 2014).

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla_\theta f_t(\theta_{t-1}))^2, \quad (3.15) \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t}, \quad (3.16)$$

onde, β_2 é a taxa de decaimento exponencial para estimativas. Com os cálculos realizados no primeiro e segundo momento, é feito a atualização dos parâmetros da rede, mostrado pela equação 3.17 (KINGMA, 2014).

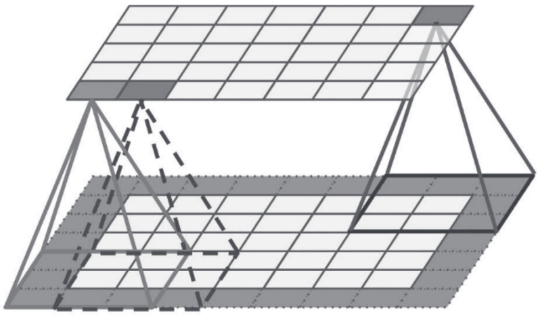
$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \quad (3.17)$$

onde, α é a taxa de aprendizagem e ϵ é um termo de estabilidade para evitar divisão por zero (KINGMA, 2014).

3.4.7 Camadas convolucionais

A CNN é um tipo específico de RNA que tem como principal objetivo, realizar tarefas relacionadas à visão computacional. Sistemas computacionais do tipo CNN exploram a estrutura espacial dos dados de entrada via operações de convolução. Conforme mostrado pela Figura 3.13, a convolução consiste na aplicação de um filtro em todas as regiões da imagem, gerando um novo mapa de características (GÉRON, 2019; LECUN *et al.*, 1998).

Figura 3.13 – Representação gráfica do filtro de convolução.



Fonte: Retirado de Géron (2019).

O filtro é uma matriz de pesos, mostrado como exemplo pela matriz 3.18, que destaca características específicas, como bordas, texturas e objetos. A aplicação de múltiplos filtros em diferentes camadas permite a extração de características cada vez mais complexas (GÉRON, 2019; LECUN *et al.*, 1998).

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.18)$$

Como mostrado pela Figura 3.13, o filtro de dimensões (3x3x1) se desloca em uma imagem de dimensões (9x7x1) com o comprimento de passo (Stride) igual a 1, ou seja, fa-

zendo operações de multiplicação e em seguida o somatório entre os pesos do filtro e os pixeis da imagem, o tamanho do mapa de características resultante possui as dimensões (7x5x1), o que leva a 35 movimentos do filtro na imagem (MEHLIG, 2021). A equação 3.19 apresenta a saída desta operação.

$$V_{ijk} = g\left(\sum_{p=1}\sum_{q=1}\sum_{r=1} w_{pqrk} x_{p+s(i-1),q+s(j-1),r} - \theta_k\right), \tag{3.19}$$

onde, g é a função de ativação, w é os pesos, x os dados de entrada, p (Altura), q (Largura) e r (Quantidade de camadas de cores) são as dimensões de uma imagem, s é o *stride*, k é o *kernel* e θ é o limite (MEHLIG, 2021).

3.4.8 Normalização em lotes

Com a aplicação das camadas de convolução, é comum na literatura a utilização de camadas de *Batch Normalization* (Em português: Normalização em lotes) para acelerar o treinamento e estabilizar a aprendizagem. Esta normalização ocorre mediante ao cálculo da média e do desvio padrão conforme a dimensão dos mini-lotes e dois parâmetros ajustáveis com o mesmo tamanho dos dados de entrada. A equação 3.20 apresenta o cálculo de normalização da saída da camada anterior (IOFFE, 2015).

$$y = \gamma\left(\frac{x - E[x]}{\sqrt{Var[x] + \epsilon}}\right) + \beta, \tag{3.20}$$

onde, x são os dados de entrada, E é a média, Var é a variância, ϵ é a constante para garantir estabilidade numérica, γ é o parâmetro ajustável para dimensionamento e β é o parâmetro ajustável para deslocamento, que por meio da retro-propagação, podem ser ajustados conforme as equações 3.21 e 3.22 (IOFFE, 2015).

$$\frac{\partial l}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial l}{\partial y_i}\left(\frac{x - E[x]}{\sqrt{Var[x] + \epsilon}}\right), \tag{3.21}$$

$$\frac{\partial l}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial l}{\partial y_i} \tag{3.22}$$

onde, l é a função de perda.

3.4.9 Função de agrupamento

A camada de *Pooling* (Em português: Agrupamento) também é comum nas implementações de CNN, onde o objetivo da sua utilização é a redução de dimensionalidade espacial mantendo as informações relevantes (GOODFELLOW; BENGIO; COURVILLE, 2016; ZHOU; CHELLAPPA, 1988). De acordo com Dumoulin e Visin (2016), uma das funções de agrupamento mais comuns é a *Max Pooling*, onde um valor máximo é definido conforme o

passo do deslocamento (*stride*) faz com que uma janela (*kernel_size*) se desloque sobre um mapa de entrada. De maneira geral, onde somente é considerado os parâmetros *kernel_size* e *stride*, a equação 3.23 apresenta o cálculo para qualquer tipo de agrupamento.

$$o = \left[\frac{i-k}{s} \right] + 1, \tag{3.23}$$

onde, i é a entrada, k é o tamanho da janela e s é o passo do deslocamento (DUMOULIN; VISIN, 2016).

3.5 Redes de Kolmogorov-Arnold

Diferente das RNAs tradicionais, baseadas no Teorema da Aproximação Universal, onde é possível aproximar qualquer função contínua desde que a função de ativação também seja contínua e não-linear (HORNIK; STINCHCOMBE; WHITE, 1989), mostrado pela equação 3.1, KANs são inspiradas no Teorema de Representação de Kolmogorov–Arnold, onde qualquer função contínua multivariável pode ser representada pelo somatório de funções unidimensionais compostas (KOLMOGOROV, 1957), mostrado pela equação 3.24.

$$f(x_1, ..., x_n) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right), \tag{3.24}$$

onde, x são as variáveis de entrada, n é a quantidade de variáveis, f é uma função contínua, Φ são funções contínuas e unidimensionais que dependem de uma única variável, ϕ é a soma entre a função base e *spline* (Curva definida por pontos de controle) , apresentada pela equação 3.25, p são as dimensões de entrada e q são as dimensões de saída (LIU *et al.*, 2024b).

$$\phi(x) = w(b(x) + spline(x)), \tag{3.25}$$

onde, b é a função base, w é o fator que controla a magnitude da função de ativação e *spline* é uma combinação linear de *B-splines* mostrado pela equação 3.26 (LIU *et al.*, 2024b).

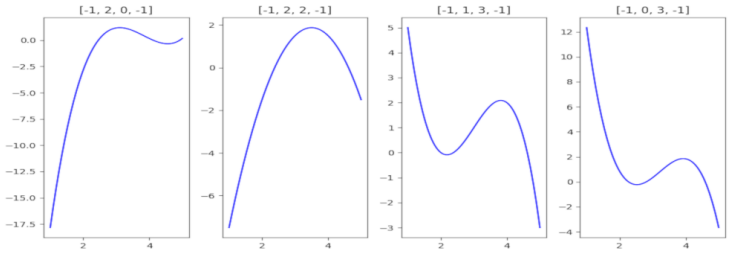
$$spline(x) = \sum_i c_i B_i(x), \tag{3.26}$$

onde, c é o parâmetro a ser ajustado e B é a função *B-spline*, onde, para funções quadráticas e cúbicas, é utilizado a função recursiva 3.27 (GORDON; RIESENFELD, 1974).

$$B_{i,m}(x) = \frac{s - x_i}{xi + m - 1 - x_i} B_{i,m-1}(s) + \frac{x_{i+m} - s}{x_{i+m} - x_{i+1}} B_{i+1,m-1}(s)^*, \tag{3.27}$$

onde, s são os nós. A escolha da função é definida pelo m, sendo 2 para funções quadráticas e 3 para funções cúbicas (GORDON; RIESENFELD, 1974). O conjunto de Figuras 3.14 apresentam gráficos com variações dos coeficientes dos polinômios cúbicos dos *B-splines*.

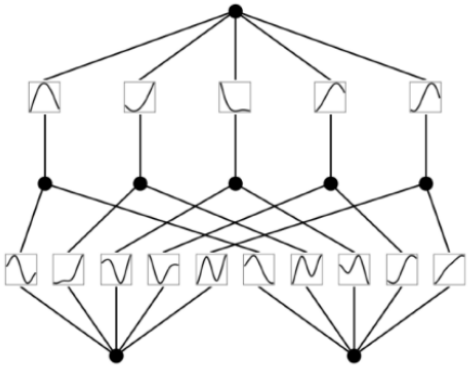
Figura 3.14 – Exemplos de *B-splines*.



Fonte: o autor.

Mesmo que a arquitetura KAN tenha uma estrutura totalmente conectada semelhante ao da RNA, como mostrado pela Figura 3.15, existem diferenças em como o cálculo da retropropagação faz a atualização dos pesos, que neste caso são funções unidimensionais (LIU *et al.*, 2024b).

Figura 3.15 – Estrutura KAN.



Fonte: Retirado de Liu et al. (2024b).

Nesta estrutura, contendo dois neurônios artificiais na camada de entrada, uma camada oculta com cinco neurônios artificiais e um neurônio artificial na camada de saída, as arestas ilustram funções de ativação ajustáveis. Para isso, neurônios artificiais não aplicam funções não-lineares no processo, em vez disso, apenas fazem o somatório dos sinais recebidos das arestas e propagam para as camadas seguintes (LIU *et al.*, 2024b).

3.6 *Efficient* KAN

Para a arquitetura *Efficient KAN*, foram feitas adaptações na implementação para melhorar a eficiência sem perdas na funcionalidade apresentada. Uma dos principais problemas da técnica KAN na sua forma original está no consumo de memória em relação à expansão da entrada de um tensor no formato (*batch_size*, *out_features*, *in_features*) para aplicação da função de ativação. Para resolver este problema, a entrada é ativada pelas funções

B-splines e depois realizadas as combinações lineares destas ativações. Outra adaptação foi a substituição da regularização L1 nas amostras de entrada para os pesos (BLEALTAN, 2024).

3.7 *Fast* KAN

Para a arquitetura *Fast KAN*, a função *B-spline* foi substituída pela função de base radial com *kernel* gaussiano para tornar o modelo mais rápido, onde cada função é centralizada em pontos diferentes no espaço de entrada, mostrado pela equação 3.28 (LI, 2024).

$$f(x) = \sum_{i=1}^{N} w_i \phi(||x - c_i||), \tag{3.28}$$

onde, N é a quantidade total de centros, w são os coeficientes ajustáveis, x é a entrada, c é o centro e ϕ é o *kernel* gaussiano, mostrado pela equação 3.29 (LI, 2024).

$$\phi(r) = exp\left(-\frac{r^2}{2h^2}\right), \tag{3.29}$$

onde, h controla a largura/dispersão da função e r é a distância radial (LI, 2024).

3.8 Pré-processamento de imagens

Devido à variedade de tamanho, qualidade, contraste e brilho das imagens do banco de dados, as técnicas utilizadas na etapa de pré-processamento tem grande relevância em relação à resposta do modelo. Para isso, técnicas de redimensionamento podem garantir uma padronização dos dados de entrada. Normalização das imagens faz com que a convergência durante o treinamento seja mais rápida ajustando os valores dos pixeis, geralmente em 0 e 1. Técnicas de aumento de dados, como, por exemplo, rotações, inversões e ajustes de brilho e contraste, aumentam a variação das imagens, evitando situações de *Overfitting*, em que o modelo tem um excelente desempenho no treinamento, mas obtém resultados ruins nos dados de teste (BALTRUSCHAT *et al.*, 2019; CHOKCHAITHANAKUL; PUNYABUKKANA; CHUANGSUWANICH, 2022; ARSOMNGERN *et al.*, 2019).

No contexto deste trabalho, as técnicas de aumento de dados não foram utilizadas somente para diversificar os dados, mas também para simular condições reais que podem acontecer durante a captura das imagens. No Capítulo 4 será apresentado como as técnicas de aumento de dados, mostrado pelo Quadro 3.5, foram implementadas.

Técnicas	Motivação
Rotação	Simular inclinações do paciente.
Deslocamento	Simular variações na posição da radiografia.
Escala	Simular diferenças no zoom.
Cisalhamento	Simular distorções causadas pelo ângulo de captura.
Brilho	Simular diferentes condições de intensidades.
Contraste	Simular diferentes condições de densidade.
Saturação	Simular diferentes condições de iluminação.
Matiz	Simular diferentes esquemas de coloração.
Ruído	Simular imperfeições causadas pelo equipamento.

Fonte: o autor.

Quadro 3.5 – Técnicas de aumento de dados utilizadas.

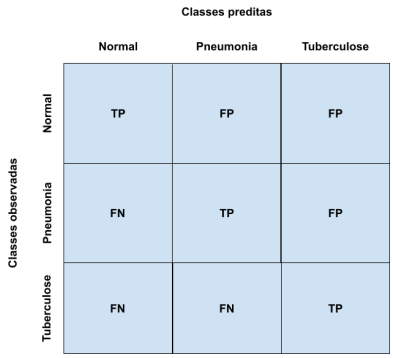
3.9 Métricas de avaliação

De acordo com Eusebi (2013), devido ao aumento do número de testes de diagnósticos nas últimas décadas, bem como de novos procedimentos, avaliações mais cuidadosas são necessárias para evitar situações perigosas, como, por exemplo, desperdício de recursos financeiros e deterioração do quadro clínico do paciente.

Devido a isto, diversas métricas de avaliação podem ser utilizadas para análise de desempenho de modelos de aprendizado de máquina do tipo supervisionado para resolução de problemas de classificação, onde é realizado um mapeamento de instâncias para classes preditas (FAWCETT, 2006), método este um dos mais utilizados para auxiliar profissionais da área da saúde no diagnóstico de doenças já conhecidas via imagens.

Ainda de acordo com Fawcett (2006), as métricas de avaliação mais comuns são Acurácia, Precisão, Sensibilidade e F1. Já a Matriz de Confusão, mostrado pela Figura 3.16, é um método que permite visualizar o desempenho do modelo em relação a possíveis confusões entre as classes que podem ocorrer durante a predição de novos dados, como por exemplo, demonstrar se possíveis erros na classificação são generalizados ou em classes específicas.

Figura 3.16 – Representação da matriz de confusão.



Fonte: Adaptado de Fawcett (2006).

Como mostrado pela Matriz de Confusão da Figura 3.16 e considerando problemas de múltiplas classes proposto neste trabalho, com as classes "Normal", "Pneumonia"e "Tuberculose", os números representados na diagonal principal são aquelas classes que foram identificadas de maneira correta, chamadas de Verdadeiros Positivos (TP - *True Positives*). Por outro lado, os valores que estão fora da diagonal principal são as classes identificadas de maneira incorreta, chamadas de Falsos Positivos (FP - *False Positives*) e Falsos Negativos (FN - *False Negatives*) (FAWCETT, 2006).

Já as métricas de avaliação Acurácia, mostrado pela equação 3.30, Precisão, mostrado pela equação 3.31, *Recall*, mostrado pela equação 3.32, e F1, mostrado pela equação 3.33, podem ser calculadas, mas para diferentes situações, visto que, o nível de balanceamento em que está um banco de dados altera o comportamento destas medidas, ou seja, a escolha das técnicas de avaliação estão diretamente ligadas com a distribuição das classes do banco de dados (FAWCETT, 2006; SAITO; REHMSMEIER, 2015).

$$\text{Acurácia} = \frac{Total\ de\ acertos}{Total\ de\ amostras} \tag{3.30}$$

A equação 3.30 é recomendada para situações em que as classes de um banco de dados estão balanceadas, visto que, a falta de balanceamento pode levar a resultados falsos devido à predição ser realizada de maneira satisfatória somente para a classe majoritária, mas ser ineficiente para as classes minoritárias (HE; GARCIA, 2009).

$$\text{Precisão(classe i)} = \frac{TP_i}{TP_i + FP_i} \tag{3.31}$$

$$\textit{Recall}\text{(classe i)} = \frac{TP_i}{TP_i + FN_i} \tag{3.32}$$

A equação 3.31 é recomendada para situações em que é desejado minimizar FP, onde FN são mais críticos. Por outro lado, a equação 3.32 é recomendada para situações em que se é desejado a minimização de FN, onde FP são mais críticos (HE; GARCIA, 2009).

$$\text{F1(classe i)} = \frac{2}{\text{Precisão}^{-1} + \text{Sensibilidade}^{-1}} \tag{3.33}$$

A equação 3.33 é recomendada para situações em que as classes do banco de dados não estão balanceadas, onde é feito o uso da média harmônica entre precisão e sensibilidade para que tanto FP quanto FN sejam minimizados (HE; GARCIA, 2009).

3.10 Avaliação de modelos e validação cruzada

Para que modelos de aprendizado de máquina que utilizam a técnica de retro-propagação façam o mapeamento da entrada em relação à saída, ou seja, aprender sobre o passado para generalizar sobre o futuro, técnicas de avaliação de modelos ou validação cruzada, como, por exemplo, *Holdout* e *KFold*, podem ser utilizadas para dividir, de maneira aleatória, um conjunto de dados que representa esta relação (HAYKIN, 2009).

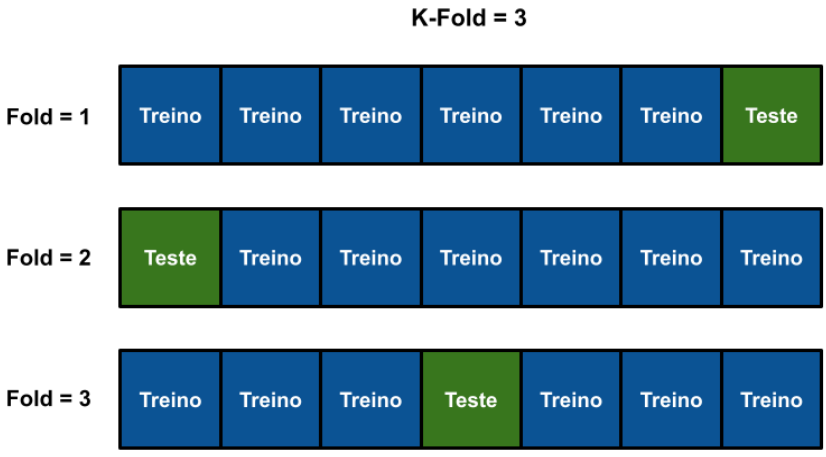
O método *Holdout* (KOHAVI, 1995) consiste em dividir, de maneira aleatória, o banco de dados em conjuntos de treinamento e teste, onde é habitual a escolha das proporções dos conjuntos de treinamento em 70% e teste em 30%, sendo possível adaptar estes valores em decorrência do tamanho do banco de dados. A Figura 3.17 ilustra este método.

Figura 3.17 – Representação gráfica do método *Holdout*.

Fonte: o autor.

Ainda de acordo com Kohavi (1995), a técnica *KFold* divide de maneira aleatória um banco de dados em subconjuntos de tamanhos aproximados, definido pelo parâmetro "k", com o intuito de avaliar o desempenho do modelo. A Figura 3.18 ilustra este método utilizando como exemplo k=3, ou seja, três subconjuntos do banco de dados são criados, sendo que, cada conjunto tem seus dados exclusivos, podendo ser definidos de maneira aleatória ou sequencial.

Figura 3.18 – Representação gráfica do método *KFold*.



Fonte: Adaptado de Haykin (2009).

Capítulo 4

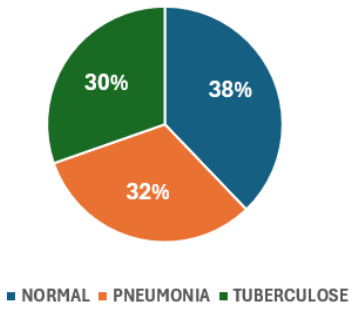
METODOLOGIA

Neste capítulo é apresentada a metodologia utilizada neste trabalho. Na Seção 4.1, são descritas as características dos bancos de dados. Na Seção 4.2, são apresentadas as bibliotecas utilizadas durante todo o processo de implementação dos modelos, bem como o *hardware* utilizado. Na Seção 4.3, são apresentadas as técnicas de pré-processamento de imagens. Na Seção 4.4, são apresentadas as etapas de treinamento-validação propostas, bem como as metodologias utilizadas para a implementação dos modelos CNN e KAN (Modelos *Efficient* e *Fast*) para cada etapa de treinamento-validação. Por fim, na Seção 4.5, são demonstradas as implementações das métricas de avaliação.

4.1 Descrição do banco de dados

O problema proposto nesta dissertação consiste na aplicação da arquitetura KAN em suas variações *Effcient* KAN e *Fast* KAN para classificar doenças pulmonares em imagens de radiografia do tórax. Para alcançar esse objetivo, foi utilizado um banco de dados contendo 13411 imagens divididas entre as classes Normal (5075 imagens), Pneumonia (4265 imagens) e Tuberculose (4071 imagens). A Figura 4.1 apresenta um gráfico em que é ilustrado a proporção entre as classes mencionadas.

Figura 4.1 – Proporção das classes do banco de dados.



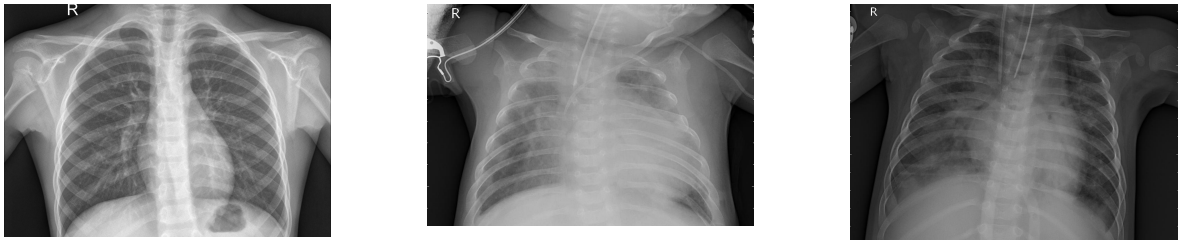
Fonte: o autor.

Este banco de dados foi construído a partir de cinco fontes diferentes, de maneira que as classes não sejam desproporcionais, ou seja, que uma classe não se sobreponha em relação a outra, evitando situações de *overfitting*.

As imagens do conjunto de dados da primeira fonte, referente ao estado natural do pulmão e pneumonia, foram adquiridas de pacientes pediátricos de um a cinco anos do Centro Médico Feminino e Infantil de Guangzhou, onde este projeto foi realizado conforme a Lei de Portabilidade e Responsabilidade de Seguros de Saúde (HIPAA - *Health Insurance Portability and Accountability Act*) dos Estados Unidos da América (USA - *United States of America*) e respeitou os princípios da Declaração de Helsinque (KERMANY *et al.*, 2018). Neste banco de dados, existem três tipos de estado em que se encontra o pulmão: condição normal, mostrado pela Figura 4.2a, pneumonia viral, mostrado pela Figura 4.2b e pneumonia bacteriana, mostrado pela Figura 4.2c (KERMANY *et al.*, 2018).

Figura 4.2 – Amostras do estado normal e pneumonia.

(a) Normal. (b) Viral. (c) Bacteriana.



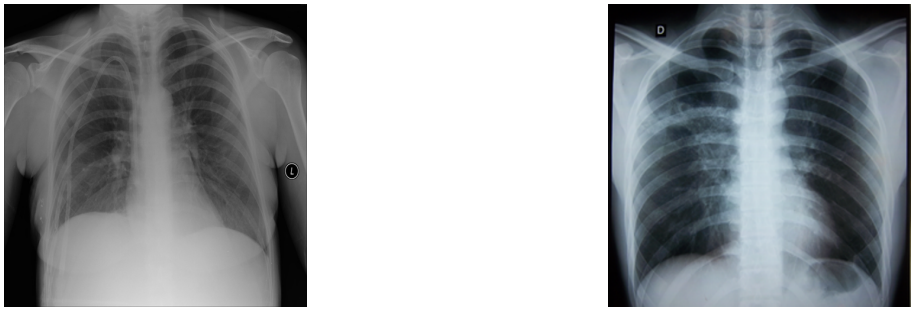
Fonte: Retirado de Kermany et al. (2018).

Devido à dificuldade de diferenciação entre os tipos de pneumonia utilizando somente imagens de radiografia do tórax, para este banco de dados, as condições viral e bacteriana foram agrupadas, portanto, serão consideradas somente as classes Normal e Pneumonia.

As imagens do conjunto de dados da segunda fonte são referentes ao estado natural do pulmão, mostrado pela Figura 4.3a e tuberculose, mostrado pela Figura 4.3b.

Figura 4.3 – Amostras do estado normal e tuberculose.

(a) Normal. (b) Tuberculose.

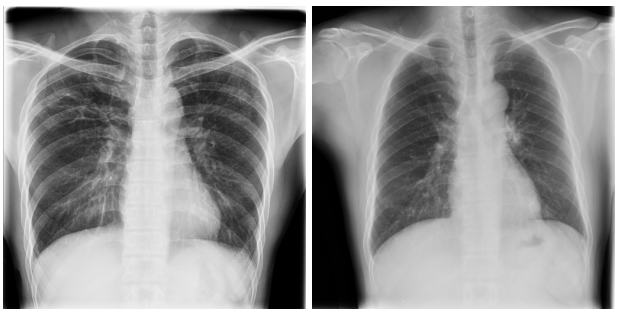


Fonte: Retirado de Rahman et al. (2020).

Estas imagens de radiografia do tórax foram adquiridas da Biblioteca Nacional de Medicina (NLM - *National Library of Medicine*) dos EUA, do conjunto de dados coletado pelo Instituto Nacional de Alergia e Doenças Infecciosas, vinculado ao Ministério da Saúde da Bielorrússia, do portal do Instituto Nacional de Alergia e Doenças Infecciosas (NIAID - *National Institute of Allergy and Infectious Diseases*) dos EUA e do banco de dados do desafio feito pela Sociedade Radiológica da América do Norte (RSNA - *Radiological Society of North America*) (RAHMAN *et al.*, 2020).

As imagens do conjunto de dados da terceira fonte são referentes a classe tuberculose, onde foram desidentificadas pelos provedores das imagens e mencionada pelos autores Liu et al. (2020) somente como "grandes hospitais". A Figura 4.4 apresenta amostras deste banco de imagens.

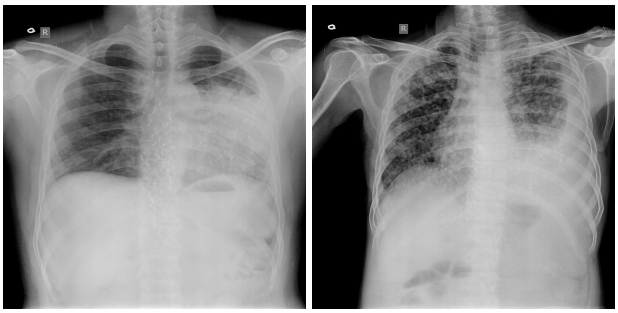
Figura 4.4 – Amostras de imagens deste conjunto



Fonte: Retirado de Liu et al. (2020).

As imagens do conjunto de dados da quarta fonte são referentes a classe tuberculose, onde foram adquiridas pelos autores Chauhan, Chauhan e Rout (2014) no Instituto Nacional de Tuberculose e Doenças Respiratórias de Nova Deli, Índia. A Figura 4.5 apresenta amostras deste banco de dados.

Figura 4.5 – Amostras de imagens deste conjunto



Fonte: Retirado de Chauhan, Chauhan e Rout (2014).

As imagens do conjunto de dados da quinta fonte são referentes a classe tuberculose, adquiridas pelos autores Kiran e Jabeen (2024) em um hospital local no Paquistão, onde a localização exata do hospital não foi revelada. A Figura 4.6 apresenta amostras deste conjunto de imagens.

Figura 4.6 – Amostras de imagens deste conjunto

Fonte: Retirado de Kiran e Jabeen (2024).

4.2 Materiais

Todas as implementações computacionais foram realizadas na linguagem *Python*, onde a biblioteca *Pytorch* foi utilizada como base para o desenvolvimento dos modelos. O Quadro 4.1 apresenta as bibliotecas utilizadas durante todo o processo de implementação, bem como suas respectivas descrições de uso.

Bibliotecas	**Descrição de uso**
torch (PASZKE *et al.*, 2019)	Aprendizado de máquina, *StepLR* e *Holdout*
torchvision (PASZKE *et al.*, 2019)	Visão computacional
time (FOUNDATION, 2024)	Medição de tempo
matplotlib (HUNTER, 2007)	Visualização de dados
numpy (HARRIS *et al.*, 2020)	Funções matemáticas
sklearn (PEDREGOSA *et al.*, 2011)	Métricas de avaliação e KFold
kan (LIU *et al.*, 2024b)	Implementação oficial KAN
efficient_kan (BLEALTAN, 2024)	Implementação eficiente KAN
fastkan (LI, 2024)	Implementação KAN para resultados mais rápidos

Fonte: o autor.

Quadro 4.1 – Bibliotecas utilizadas.

Para realizar o treinamento de técnicas de aprendizado de máquina, o *hardware* tem grande impacto na velocidade em que ocorrem os cálculos da retro-propagação. Portanto, o Quadro 4.2 apresenta o *hardware* utilizando nas aplicações dos modelos.

Hardware	**Descrição**
CPU	i5-3470
GPU	NVIDIA GTX 1650 4GB
RAM	16 GB
SSD	256,1 GB

Fonte: o autor.

Quadro 4.2 – *Hardware* utilizado.

4.3 Implementação do pré-processamento das imagens

O Código 4.1 apresenta a implementação das técnicas de pré-processamento das imagens do banco de dados completo, onde o módulo *transfoms.Compose()* da biblioteca *torchvision*, localizada na linha 1, foi utilizada para agrupar as transformações escolhidas.

Código 4.1 – Implementação das técnicas de aumento de dados.

```
1  transform_dataset = transforms.Compose(
2      [transforms.Resize(size=(164,164)), # ou (224,224)
3       transforms.RandomRotation(degrees=10),
4       transforms.RandomAffine(translate=(0.1, 0.1), scale=(0.9, 1.1),
           shear=10),
5       transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation
           =0.1, hue=0.1),
6       transforms.ToTensor(),
7       transforms.Lambda(lambda x: torch.clamp(x + torch.randn_like(x) *
           0.05, 0., 1.)),
8       transforms.Normalize(mean=[0.5], std=[0.5])
9      ]
10 )
```

Como mostrado pelo Código 4.1, foram utilizadas sete funções de transformação, sendo a primeira *transforms.Resize()*, na linha 2, responsável pela padronização do redimensionamento das imagens nas dimensões 164x164 ou 224x224. A segunda função *transforms.RandomRotation()*, na linha 3, foi utilizada para rotacionar as imagens em ângulos de 10 graus, de maneira aleatória. A terceira função *transforms.RandomAffine()*, na linha 4, foi utilizada para aplicar três transformações nas imagens, sendo a primeira, definida pelo parâmetro *translate*, que desloca a imagem até 10% do seu tamanho, tanto na horizontal quanto na vertical. O segundo parâmetro *scale*, escalona, de maneira aleatória, entre 90% de redução e 110% de aumento das imagens em relação ao seu tamanho. Por fim, no parâmetro *shear* é definido o ângulo de cisalhamento em ±10 graus. Na quarta função *transforms.ColorJitter()*, na linha 5, quatro parâmetros foram utilizados para aplicar variações nas cores das imagens, sendo o primeiro definido por *brightness*, que ajusta o brilho da imagem, de maneira aleatória, em um intervalo de ±20%. O parâmetro *contrast*, altera o contraste da imagem, de maneira aleatória, em um intervalo de ±20%. O parâmetro *saturation*, ajusta a intensidade das cores, de maneira aleatória, em um intervalo de ±10%. Por fim, o parâmetro *hue*, altera o matiz da imagem, de maneira aleatória, em um intervalo de ±10%. A quinta função *transforms.ToTensor()*, na linha 6, visa converter as imagens do formato *array* para tensor. A sexta função personalizada *transforms.Lambda()*, na linha sete, simula ruídos que podem ocorrer no momento da captura das imagens. Utilizando a função *torch.randn_like(x) * 0.05*, um tensor do mesmo formato da imagem é criado, onde cada pi-

xel receberá um valor de ruído diferente. Em seguida é feito uma multiplicação com o valor 0.05 para ajustar a intensidade do ruído. Por fim este ruído somado com a imagem passa pela função *torch.clamp*, garantindo que os valores dos tensores estejam entre 0 e 1. A última função *Normalize*, na linha oito, visa normalizar em 0,5 os valores dos pixeis utilizando os parâmetros *mean* (Média) e *std* (Desvio padrão).

4.4 Etapas de treinamento-validação

A metodologia proposta para as etapas de treinamento e validação para classificação de doenças pulmonares utilizando imagens de radiografia do tórax estão envoltas entre três etapas, onde os modelos CNN e KAN (*Efficient* e *Fast*) foram empregados em todas as etapas. Na primeira etapa, referente a testes iniciais, foi utilizado um conjunto amostral reduzido contendo 1000 imagens selecionadas a partir do banco de dados completo, para verificar com mais velocidade, possíveis erros que podem ocorrer na sintaxe da implementação, se os dados estão sendo carregados de maneira correto com os seus respectivos rótulos e possibilitando uma quantidade maior de implementações para testagem de parâmetros. O método *Holdout* foi empregado para dividir o conjunto de amostras em subconjuntos de treinamento e teste, respectivamente, 70% e 30% em relação ao total do conjunto amostral, ou seja, 700 imagens foram utilizadas no treinamento e 300 imagens nos testes, mantendo as classes balanceadas. O Código 4.2 apresenta a implementação deste método de avaliação de modelos por meio da função *random_split()* da biblioteca *torch*.

Código 4.2 – Implementação do método de avaliação *Holdout*.

```
1 dataset = datasets.ImageFolder(dataset_path, transform =
    transform_dataset)
2
3 train_size = int(0.7 * len(dataset))
4 test_size = len(dataset) - train_size
5 train_dataset, test_dataset = random_split(dataset, [train_size,
    test_size])
```

No Código 4.2, a linha 1 representa a localização da pasta em que se encontra o conjunto amostral, onde a função *datasets.ImageFolder()*, da biblioteca *torchvision*, foi utilizada com os parâmetros de localização, representado pela variável *dataset_path* e pelo parâmetro da aplicação das técnicas de pré-processamento das imagens, representado pela variável *transform*. As linhas 3 e 4 representam as definições das porcentagens em que o conjunto será dividido. Na linha 3, o tamanho do conjunto amostral (Função *len()*, retornando o valor 1000) foi multiplicado por 0.7, resultando no valor 700 e atribuído a variável *train_size*. Na linha 4, foi realizada uma subtração entre o tamanho do conjunto amostral com *train_size*, resultando no valor 300 e atribuído a variável *test_size*. Por fim, na linha 5,

a função *random_split()* foi utilizada para dividir, de maneira aleatória, entre o subconjunto de treinamento, representado pela variável *train_dataset* e teste, representado pela variável *test_dataset*, por meio das variáveis descritas nas linhas 1 e 2, como também pela atribuição ao conjunto amostral, representado pela variável *dataset*.

Para a segunda etapa, referente à modificação do conjunto de dados, manteve-se a metodologia utilizada na aplicação da técnica *Holdout* da primeira etapa, mas com a utilização do banco de dados completo contendo 13411 imagens. Portanto, o subconjunto de treinamento totalizou 9387 imagens e o subconjunto de teste, 4024 imagens.

Para a terceira etapa, referente à modificação das técnicas de separação dos dados, manteve-se o banco de dados completo contendo 13411 imagens, mas com alterações na maneira em que a divisão dos dados são feitas, onde a técnica de avaliação de modelos *Holdout* foi substituída pela técnica de validação cruzada *KFold*. O Código 4.3 apresenta a implementação deste método.

Código 4.3 – Implementação da técnica de validação cruzada *KFold*.

```
1 k_folds = 3
2 kf = KFold(n_splits=k_folds, shuffle=True, random_state=42)
3 for fold, (train_idx, test_idx) in enumerate(kf.split(dataset)):
4     ###Implementacao do modelo###
```

No Código 4.3, a função *KFold* da biblioteca *sklearn*, localizada na linha 2 e atribuída a variável *kf*, foi utilizada para criar múltiplos subconjuntos por meio dos parâmetros: *n_splits*, que define a quantidade de subconjuntos, definido pelo valor 3 e atribuído a variável *k_folds* na linha 1; *shuffle*, que define a maneira em que é feita as escolhas dos dados de treino e teste, sendo *True* para divisão aleatória e *False* para divisão sequencial e *random_state*, que foi utilizada para garantir reprodutividade das aplicações dos modelos, ou seja, controlar a maneira em que os dados são divididos de maneira aleatória, visto que, o parâmetro *shuffle* foi atribuído como *True*. O laço de repetição, localizado na linha 3, retorna os índices dos dados divididos em treino (*train_idx*) e teste (*test_idx*) para cada iteração.

4.4.1 Implementações dos modelos na primeira etapa

Nesta primeira etapa de testes iniciais, o carregamento em mini-lotes foi realizado de maneira idêntica para todos os modelos, mostrado pelo Código 4.4.

Código 4.4 – Implementação do carregamento do conjunto em mini-lotes.

```
1 train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
2 test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)
```

No Código 4.4, o carregamento dos dados foi realizado utilizando a função *DataLoader* da biblioteca *torch*, onde, na linha 1, foi atribuído ao conjunto de treinamento e na linha 2, foi atribuído ao conjunto de teste. Para esta função, três parâmetros foram utilizados, sendo o primeiro referente aos conjuntos definidos no momento da divisão dos conjuntos entre treino e teste, o segundo, referente ao tamanho dos mini-lotes, definido com o valor 16 e o terceiro, que realiza o embaralhamento dos dados, onde na linha 1, para os dados de treinamento, foi definido com verdadeiro e na linha 2, para os dados de teste, foi definido como falso.

Para o modelo CNN, cinco testes foram realizados, com variações na arquitetura, no valor inicial da taxa de aprendizagem e na maneira em que essa taxa é atualizada. A primeira implementação foi realizada com a arquitetura mostrada pelo Código 4.5.

No Código 4.5, a arquitetura do modelo CNN foi realizada utilizando a função *nn.Sequential* da biblioteca *torch*, onde o momento de execução é realizado de maneira sequencial, conforme a posição da operação. As três camadas convolucionais, representadas pela função *nn.Conv2d()* da biblioteca *torch*, nas linhas 2, 7 e 12, foram utilizadas com os parâmetros *in_channels* (Canais de entrada), *out_channels* (Canais de saída), *kernel_size* (Tamanho do *kernel*), *stride* (Comprimento do passo) e *padding* (Controla a quantidade de preenchimento ao redor da borda da entrada). A Tabela 4.1 apresenta os valores atribuídos para cada parâmetro das três camadas convolucionais.

Tabela 4.1 – Camadas convolucionais.

Parêmetros	**Primeira camada**	**Segunda camada**	**Terceira camada**
in_channels	3	32	64
out_channels	32	64	128
kernel_size	3	3	3
stride	1	1	1
padding	1	1	1

Em relação à normalização dos mapas de características, representadas pela função *nn.BatchNorm2d()* da biblioteca *torch*, localizadas nas linhas 3, 8 e 13, recebem o valor do parâmetro *out_channels* de cada camada convolucional anterior. Para representar as relações não-lineares, a função de ativação *ReLU* foi utilizada seis vezes, nas linhas 4, 9, 14, 21, 25 e 29. A redução das dimensões dos mapas de características normalizadas, representado pela função *nn.MaxPool2d()*, foram realizadas três vezes, nas linhas 5, 10 e 15, onde, para estas camadas, os parâmetros foram definidos como, *kernel_size*=2, *stride*=2 e *padding*=0.

Após aplicado a função *nn.Flatten()* para achatar a imagem em um vetor unidimensional, mostrado na linha 17, foram utilizadas quatro funções lineares totalmente conectadas, *nn.Linear()*, nas linhas 19, 23, 27 e 31. A Tabela 4.2 apresenta as distribuições dos neurônios artificiais, onde o valor de entrada (128*28*28 = 100352) da primeira camada corresponde a saída resultante da última operação de redimensionamento após ser achatada, onde 128

Código 4.5 – Primeira arquitetura do modelo CNN.

```
model = nn.Sequential(
    nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1),
    nn.BatchNorm2d(32),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2, padding=0),

    nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
    nn.BatchNorm2d(64),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2, padding=0),

    nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
    nn.BatchNorm2d(128),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2, padding=0),

    nn.Flatten(),

    nn.Linear(128 * 28 * 28, 256),
    nn.Dropout(p=0.3),
    nn.ReLU(),

    nn.Linear(256, 128),
    nn.Dropout(p=0.3),
    nn.ReLU(),

    nn.Linear(128, 64),
    nn.Dropout(p=0.3),
    nn.ReLU(),

    nn.Linear(64, 3)
)
```

representa a quantidade de mapas de características da últimas camada convolucional normalizada e 28x28, representa a dimensão resultante das divisões realizadas a cada redução em relação ao tamanho original da imagem original, com tamanho 224x224.

Tabela 4.2 – Distribuição de neurônios artificiais do primeiro teste do modelo CNN.

Parâmetros	Primeira camada	Segunda camada	Terceira camada	Camada de saída
Entrada	128*28*28	256	128	64
Saída	256	128	64	3

Para prevenir situações de *overfitting*, a função *nn.Dropout()* foi utilizada três vezes, nas linhas 20, 24 e 28, com probabilidade de desativação de cada neurônio artificial em 30%

de maneira aleatória. O Quadro 4.3 apresenta os parâmetros utilizados no primeiro treinamento para o modelo CNN.

Parâmetros	**Descrição**
num_epoch	250
learning_rate	0,01
optimizer	optim.Adam()
loss_fn	nn.CrossEntropyLoss()

Fonte: o autor.

Quadro 4.3 – Parâmetros do primeiro treinamento do modelo CNN.

A realização do treinamento ocorre mediante a dois laços de repetição, sendo o primeiro, responsável pelo avanço das épocas (*num_epoch*), mostrado pelo Código 4.6, e o segundo, responsável pelo treinamento conforme os conjuntos dos mini-lotes, mostrado pelo Código 4.7.

Código 4.6 – Laço de repetição das épocas do treinamento.

```
1 for epoch in range(num_epoch):
2     model.train()
3     running_train_loss = 0.0
4
5     # Laco de repeticao dos mini-lotes
6
7     train_loss = running_train_loss / len(train_loader)
8     _, predicted = torch.max(outputs.data, 1)
```

No Código 4.6, a função *model.train()*, na linha 2, representa que o modelo foi colocado em modo de treinamento, ou seja, faz com que as camadas mantenham o seu comportamento normal, utilizando as estatísticas do lote atual. Para os cálculos realizados nas linhas 7 (Perda mediante a divisão do acúmulo das perdas dos lotes pelo tamanho destes lotes) e 8 (Predição mediante a utilização da função *torch.max()* para obtenção dos índices das classes de maior probabilidade a partir das saídas do modelo), são utilizados parâmetros calculados no Código 4.7, que na implementação original está dentro no laço de repetição do Código 4.6, como mostrado pelo comentário na linha 5.

Para cada iteração mediante ao valor definido de mini-lotes, mostrado pelo laço de repetição do Código 4.7, são realizadas as operações apresentadas pelo Quadro 4.4.

Para validar o treinamento, o Código 4.8 foi implementado em um ambiente de avaliação, pelo uso da função *model.eval()*, na linha 1, onde é utilizado as estatísticas acumuladas durante o treinamento. Na linha dois, a função *torch.no_grad()* faz com que os cálculos dos gradientes sejam desativados, visto que nesta etapa não existe a necessidade de atualizar os pesos.

Código 4.7 – Laço de repetição dos mini-lotes do treinamento.

```
1 for inputs, labels in train_loader:
2     inputs = inputs.to(device)
3     labels = labels.to(device)
4     optimizer.zero_grad()
5     outputs = model(inputs)
6     loss = loss_fn(outputs, labels)
7     loss.backward()
8     optimizer.step()
9     running_train_loss += loss.item()
```

Linhas	Descrições
2 e 3	Valores de entrada e rótulos movidos para GPU.
4	Gradientes zerados para evitar acumulação.
5	Predições a partir da aplicação do modelo a cada mini-lote.
6	Cálculo da perda entre as predições e os rótulos.
7	Cálculo dos gradientes da perda (Retro-propagação).
8	Atualização dos pesos com base na retro-propagação.
9	Acumula a perda ao longo das iterações.

Fonte: o autor.

Quadro 4.4 – Operações de treinamento.

Código 4.8 – Implementação para validação do modelo CNN.

```
1 model.eval()
2 with torch.no_grad():
3     for images, labels in testloader:
4         images, labels = images.to(device), labels.to(device)
5         outputs = model(images)
6         _, predicted = torch.max(outputs.data, 1)
```

Na segunda implementação para o modelo CNN, a única modificação realizada foi no parâmetro *learning_rate*, com valor inicial em 0.1 e atualização dinâmica realizada pela técnica *StepLR* da biblioteca *torch*. O Código 4.9 apresenta a implementação desta técnica.

No Código 4.9, a atualização da taxa de aprendizagem é realizada pela função *lr_scheduler.StepLR(* para definir os parâmetros de como será feita esta atualização, na linha 1, e pela função *scheduler.step()* onde será feita a atualização, na linha 8. A escolha deste método se justifica devido a simplicidade, onde o valor inicial da taxa de aprendizagem é multiplicado por 0.1, mostrado pelo parâmetro *gamma*, a cada 50 épocas, mostrado pelo parâmetro *step_size*.

Na terceira implementação para o modelo CNN, a taxa de aprendizagem foi definida de maneira estática, com valor reduzido em 0.001. O restante da aplicação se manteve idêntica ao que foi apresentado na primeira implementação do modelo CNN.

Código 4.9 – Implementação da técnica *StepLR*.

```
scheduler=optim.lr_scheduler.StepLR(optimizer,step_size=50,gamma=0.1)
for epoch in range(num_epoch):
    model.train()
    running_train_loss = 0.0

    # Laco de repeticao dos mini-lotes

    scheduler.step()
    train_loss = running_train_loss / len(train_loader)
    _, predicted = torch.max(outputs.data, 1)
```

Na quarta implementação para o modelo CNN, novamente a taxa de aprendizagem foi o único parâmetro que passou por mudanças, onde o valor inicial foi mantido em 0.001, mas com a alteração para atualização dinâmica utilizando a função *lr_scheduler.StepLR()* com os mesmos parâmetros definidos na segunda implementação, mostrado pelo Código 4.9.

Na quinta implementação para o modelo CNN, foram adicionadas mais camadas na arquitetura, sendo elas: *Conv2d(128, 256)*, *BatchNorm2d(256)* e *ReLU()*, mostrado pelo Código 4.10. Todos os parâmetros restantes foram mantidos, ou seja, número de épocas em 250, taxa de aprendizagem dinâmica em 0.001, otimizador *Adam* e função de perda *CrossEntropyLoss*.

Código 4.10 – Camadas adicionadas na 5^a implementação do modelo CNN.

```
nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1),
nn.BatchNorm2d(256),
nn.ReLU(),
```

Devido à adição das camadas demonstradas pelo Código 4.10, foram necessárias algumas modificações em relação às camadas lineares, mostrado pelo Código 4.11, visto que a saída da camada convolucional normalizada não corresponde ao que foi apresentado pelo Código 4.5.

Código 4.11 – Camadas lineares modificadas para a 5^a implementação do modelo CNN.

```
nn.Linear(256*28*28, 256),
...
nn.Linear(32, 3)
```

A camada linear apresentada no Código 4.11, recebe os 256 mapas de características da camada convolucional normalizada apresentada no Código 4.10, onde permaneceu inalterado as dimensões 28*28, visto que as camadas de redimensionamento não foram modi-

ficadas. Na linha 3, apresenta a camada de saída modificada devido às alterações realizadas na arquitetura por meio das camadas *Linear(64, 32)*, *Dropout(p=0.3)* e *ReLU()*, mostradas pelo Código 4.12, ou seja, a última camada foi modificada para receber a saída da camada linear adicionada.

Código 4.12 – Camadas adicionadas na 5^a implementação do modelo CNN.

```
1 nn.Linear(64, 32),
2 nn.Dropout(p=0.3),
3 nn.ReLU(),
```

O Código 4.13 apresenta a implementação completa desta arquitetura.

Código 4.13 – Quinta implementação do modelo CNN.

```
1 nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1), #Mantido
2 nn.BatchNorm2d(32), #Mantido
3 nn.ReLU(), #Mantido
4 nn.MaxPool2d(kernel_size=2, stride=2, padding=0), #Mantido
5 nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1), #Mantido
6 nn.BatchNorm2d(64), #Mantido
7 nn.ReLU(), #Mantido
8 nn.MaxPool2d(kernel_size=2, stride=2, padding=0), #Mantido
9 nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1), #Mantido
10 nn.BatchNorm2d(128), #Mantido
11 nn.ReLU(), #Mantido
12 nn.MaxPool2d(kernel_size=2, stride=2, padding=0), #Mantido
13 nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1), #Adicionado
14 nn.BatchNorm2d(256), #Adicionado
15 nn.ReLU(), #Adicionado
16 nn.Flatten(), #Mantido
17 nn.Linear(256 * 28 * 28, 256), #Alterado
18 nn.Dropout(p=0.3), #Mantido
19 nn.ReLU(), #Mantido
20 nn.Linear(256, 128), #Mantido
21 nn.Dropout(p=0.3), #Mantido
22 nn.ReLU(), #Mantido
23 nn.Linear(128, 64), #Mantido
24 nn.Dropout(p=0.3), #Mantido
25 nn.ReLU(), #Mantido
26 nn.Linear(64, 32), #Adicionado
27 nn.Dropout(p=0.3), #Adicionado
28 nn.ReLU(), #Adicionado
29 nn.Linear(32, 3) #Alterado
```

Para o modelo *Efficient* KAN, três implementações foram realizadas, onde a maioria das técnicas de pré-processamento de imagens foram mantidas inalteradas, menos a técnica de redimensionamento das imagens, que, para o modelo CNN foi utilizado o tamanho

224x224, mas para os modelos baseados na arquitetura KAN (*Efficient* e *Fast*), foi utilizado o tamanho 164x164. O Quadro 4.5 apresenta os parâmetros utilizados para os três modelos.

Implementação	**1**^{a}	**2**^{a}	**3**^{a}
Taxa de aprendizagem	0.001	0.001	0.001
Tipo de taxa de aprendizagem	Estática	Dinâmica	Dinâmica
Camadas	4	4	5
Otimizador	*Adam*	*Adam*	*Adam*
Função de perda	*CrossEntropyLoss*	*CrossEntropyLoss*	*CrossEntropyLoss*
Épocas	250	250	250

Fonte: o autor.

Quadro 4.5 – Operações de treinamento para o modelo *Efficient* KAN.

O Código 4.14 apresenta a primeira implementação para o modelo *Efficient* KAN, com 4 camadas, $grid\_size$=5 (Número de intervalos da grade) e $spline\_order$ = 3 (Ordem do polinômio - Função cúbica), sendo estes dois últimos parâmetros válidos para todas as implementações do modelo *Efficient* KAN deste trabalho.

Código 4.14 – Primeira implementação para o modelo *Efficient* KAN.

```
1 model = KAN([164*164*3, 164, 64, 3])
```

O Código 4.14, demonstra o uso do *framework KAN()* em que o primeiro parâmetro corresponde às dimensões das imagens, ou seja, definido pelo redimensionamento na parte do pré-processamento das imagens. A última camada corresponde a saída do modelo, visto que são 3 rótulos a serem classificados (Normal, Pneumonia e Tuberculose). Todos os valores entre as camadas de entrada e saída representam as camadas ocultas. Portanto, serão 164x164x3 neurônios artificiais na camada de entrada, onde cada neurônio receberá um pixel da imagem, 2 camadas ocultas com, respectivamente, 164 e 64 neurônios, e 3 neurônios para camada de saída, representando cada classe a ser identificada.

Como mostrado pela Tabela 4.5, para a segunda implementação utilizando o modelo *Efficient* KAN, somente a maneira em que é feita a atualização da taxa de aprendizagem é que foi alterado, em comparação a primeira implementação. Para a terceira e última implementação utilizando este modelo, comparando com a segunda implementação, somente a quantidade de camadas é que passou por alterações, onde o Código 4.15 apresenta a segunda arquitetura a ser utilizada para este modelo.

Código 4.15 – Terceira implementação para o modelo *Efficient* KAN.

```
1 model = KAN([164*164*3, 164, 64, 32, 3])
```

Para o modelo *Fast* KAN, três implementações foram realizadas, mantendo todas as técnicas de pré-processamento de imagens e as operações de treinamento utilizadas nas

implementações do modelo *Efficient* KAN, mostradas pelo Quadro 4.5. Mas, como descrito por Li (2024), foi utilizando 8 centros no cálculo das Funções de Base Radial, com também foram utilizadas camadas de normalização na camada de entrada e nas camadas ocultas para evitar que os domínios entre as Funções de Base Radial e os valores de entrada sejam diferentes. O Código 4.16 apresenta a arquitetura para as duas primeiras implementações, enquanto o Código 4.17 apresenta a arquitetura para a terceira e última implementação.

Código 4.16 – 1^a e 2^a arquitetura para o modelo *Fast* KAN.

```
model = KAN([164*164*3, 164, 64, 3])
```

Código 4.17 – 3^a arquitetura para o modelo *Fast* KAN.

```
model = KAN([164*164*3, 164, 64, 32, 3])
```

Visto que o modelo *Fast* KAN é baseado no *Efficient* KAN, a maneira em que é implementado os Códigos 4.16 e 4.17 é idêntica aos Códigos 4.14 e 4.15, do modelo *Efficient* KAN, onde a diferença está na biblioteca utilizada, onde a técnica *Efficient* KAN é aplicada por meio da biblioteca *efficient_kan*, e a técnica *Fast* KAN é aplicada por meio da biblioteca *fastkan*.

4.4.2 Implementações dos modelos na segunda etapa

Para a segunda etapa, o banco de dados foi a principal mudança realizada, onde o conjunto amostral contendo 1000 imagens, utilizado na primeira etapa, foi para um banco de dados completo contendo 13411 imagens. Para os modelos CNN, *Efficient* KAN e *Fast* KAN, dois experimentos foram realizados para cada modelo considerando o uso da técnica *StepLR* para atualização da taxa de aprendizagem. O Quadro 4.6 apresenta a arquitetura e os parâmetros para cada modelo.

Implementações	**CNN**		***Efficient* KAN**		***Fast* KAN**	
	$\mathbf{1}^a$	$\mathbf{2}^a$	$\mathbf{1}^a$	$\mathbf{2}^a$	$\mathbf{1}^a$	$\mathbf{2}^a$
Arquitetura	Código 4.13	Código 4.13	Código 4.15	Código 4.15	Código 4.17	Código 4.17
Taxa de aprendizagem	0.001	0.001	0.001	0.001	0.001	0.001
Tipo de atualização	Estático	*StepLR*	Estático	*StepLR*	Estático	*StepLR*
Otimizador	*Adam*	*Adam*	*Adam*	*Adam*	*Adam*	*Adam*
Função de perda	*CELoss*	*CELoss*	*CELoss*	*CELoss*	*CELoss*	*CELoss*
Épocas	250	250	250	250	250	250

Fonte: o autor.

Quadro 4.6 – Arquiteturas e parâmetros para os modelos da 2^a etapa.

4.4.3 Implementações dos modelos na terceira etapa

Para a terceira etapa, serão realizadas três implementações, uma para cada modelo, onde se manteve o banco de dados completo e as arquiteturas das últimas aplicações de cada modelo da segunda etapa, bem como os parâmetros apresentados pelo Quadro 4.6, mas desconsiderando o uso da técnica *StepLR* para atualização dinâmica da taxa de aprendizagem.

O destaque desta etapa está no uso da técnica de validação cruzada *KFold*, onde a sua aplicação básica foi apresentada no Código 4.3. No entanto, algumas medidas devem ser realizadas para evitar erros na aplicação desta técnica, visto que, para garantir a reprodutibilidade ao iniciar um novo *Fold* é necessário reiniciar a rede. O Código 4.18 apresenta a implementação destas medidas.

Código 4.18 – Medidas para reiniciar os pesos a cada *Fold*.

```
1 torch.backends.cudnn.deterministic = True
2 torch.backends.cudnn.benchmark = False
3
4 def reset_weights(m):
5     if isinstance(m, (nn.Conv2d, nn.Linear)):
6         m.reset_parameters()
```

Para evitar situações de estados ocultos na GPU, que podem interferir na reprodutibilidade entre os *folds*, na linha 1, a função *torch.backends.cudnn.deterministic* definida como verdadeiro, força a realização de operações determinísticas, garantindo que a mesma entrada sempre produzirá a mesma saída, evitando inconsistências entre os *Folds*. Já na linha 2, a função *torch.backends.cudnn.benchmark* definida como falso, usa algoritmos fixos para o tamanho dos dados de entrada, fazendo com que cada *Fold* seja iniciado com os mesmos cálculos. Na linha 4, a função *reset_weights()* tem com objetivo reinicializar os parâmetros da rede por meio da função *isinstance()*, que verifica se o objeto pertence a uma determinada classe, ou seja, se *m* é uma camada treinável (Conv2d e Linear), a função *reset_parameters()* reinicia os pesos destas camadas. O Código 4.19 demonstra as modificações realizadas.

Como já mencionado anteriormente, foram utilizados 3 *Folds* nas implementações desta etapa, portanto, como mostrado no Código 4.19, na linha 3, a função *reset_weights* é aplicada ao modelo em cada *Fold* para redefinir os pesos. Nas linhas 4 e 5, são inicializados a cada *Fold* o otimizador *Adam* e a função de perda *CrossEntropyLoss*. Nas linhas 6 e 7, a função *SubsetRandomSample()*, da biblioteca *torch*, foi utilizada para criar os subconjuntos de treinamento e teste, de maneira aleatória, em relação aos índices para cada *Fold*.

Os códigos completos podem ser encontrados no repositório do *GitHub* localizado no Apêndice A.

Código 4.19 – Modificações realizadas para a técnica *KFold*.

```
for fold, (train_idx, test_idx) in enumerate(kf.split(dataset)):
    model = cnn_model().to(device)
    model.apply(reset_weights)
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)
    loss_fn = nn.CrossEntropyLoss()
    train_sampler = SubsetRandomSampler(train_idx)
    test_sampler = SubsetRandomSampler(test_idx)
    trainloader=DataLoader(dataset,batch_size=16,sampler=train_sampler)
    testloader=DataLoader(dataset,batch_size=16,sampler=test_sampler)
    # Lacos de repeticao das epocas
        # Lacos de repeticao dos mini-lotes
```

4.5 Implementação das métricas de avaliação

Para validação dos modelos, as métricas de avaliação já conhecidas na literatura, Matriz de Confusão, Precisão, *Recall*, Acurácia e pontuação F1 foram utilizadas para comparar o desempenho dos modelos. O Código 4.20 apresenta a implementação da Matriz de Confusão utilizando a biblioteca *sklearn*.

Código 4.20 – Implementação da Matriz de Confusão.

```
labels = labels.cpu().detach().numpy()
predicted = predicted.cpu().detach().numpy()
classes = ['Normal', 'Pneumonia', 'Tuberculose']
cm = confusion_matrix(labels, predicted)
disp = ConfusionMatrixDisplay(cm, classes)
disp.plot(cmap=plt.cm.Blues)
plt.xlabel('Rotulo_previsto')
plt.ylabel('Rotulo_verdadeiro')
```

No Código 4.20, a função *confusion_matrix*, bem como as outras métricas de avaliação, não calculam dados que estão alocados no dispositivo GPU, portanto, nas linhas 1 e 2, foram utilizadas as funções *cpu().detach().numpy()* para realocar estes dados para o dispositivo CPU. Por outro lado, o Código 4.21 apresenta as implementações das métricas de avaliação, Acurácia, Precisão, *Recall* e pontuação F1.

Código 4.21 – Implementação das métricas de avaliação.

```
acc = accuracy_score(labels, predicted)
precision = precision_score(labels, predicted, average='weighted')
recall = recall_score(labels, predicted, average='weighted')
f1 = f1_score(labels, predicted, average='weighted')
```

Capítulo 5

RESULTADOS E DISCUSSÕES

Na 1^a etapa, foi utilizado o conjunto amostral contendo 1000 imagens, mostrado pela Tabela 5.1, que apresenta as descrições de uso para cada implementação dos modelos CNN, *Efficient* KAN e *Fast* KAN, para as classes, Normal, com 340 imagens, Pneumonia, com 330 imagens e Tuberculose, com 330 imagens. Por meio do método *Holdout*, de maneira aleatória, 70% das imagens foram selecionadas para o treinamento e 30% para os dados de teste.

Tabela 5.1 – Distribuição das classes para a 1^a etapa.

Implementação	Tipo	Normal	Pneumonia	Tuberculose
1^a - CNN	Treinamento	245 (35%)	221 (31.6%)	234 (33.4%)
	Teste	95 (31.7%)	109 (36.3%)	96 (32%)
2^a - CNN	Treinamento	254 (36.3%)	224 (32%)	222 (31.7%)
	Teste	86 (28.7%)	106 (35.3%)	108 (36%)
3^a - CNN	Treinamento	245 (35%)	223 (31.9%)	232 (33.1%)
	Teste	95 (31.7%)	107 (35.7%)	98 (32.7%)
4^a - CNN	Treinamento	242 (34.6%)	237 (39.9%)	221 (31.6%)
	Teste	98 (32.7%)	93 (31%)	109 (36.3%)
5^a - CNN	Treinamento	240 (34.3%)	221 (31.6%)	239 (34.1%)
	Teste	100 (33.3%)	109 (36.3%)	91 (30.3%)
1^a - EKAN	Treinamento	242 (34.6%)	232 (33.1%)	226 (32.3%)
	Teste	98 (32.7%)	98 (32.7%)	104 (34.7%)
2^a - EKAN	Treinamento	240 (34.3%)	227 (32.4%)	233 (33.3%)
	Teste	100 (33.3%)	103 (34.3%)	97 (32.3%)
3^a - EKAN	Treinamento	229 (32.7%)	248 (35.4%)	223 (31.9%)
	Teste	111 (37%)	82 (27.3%)	107 (35.7%)
1^a - FKAN	Treinamento	241 (34.4%)	226 (32.3%)	233 (33.3%)
	Teste	99 (33%)	104 (34.7%)	97 (32.3%)
2^a - FKAN	Treinamento	231 (33%)	233 (33.3%)	236 (33.7%)
	Teste	109 (36.3%)	97 (32.3%)	94 (31.3%)
3^a - FKAN	Treinamento	238 (34%)	229 (32.7%)	233 (33.3%)
	Teste	102 (34%)	101 (33.7%)	97 (32.3%)

Fonte: o autor.

Para a segunda etapa, experimentos foram realizados utilizando o banco de dados completo contendo 13411 imagens. A Tabela 5.2, apresenta as descrições deste banco de dados para cada implementação dos modelos CNN, *Efficient* KAN e *Fast* KAN, para as classes, Normal (5075 imagens), Pneumonia (4265 imagens) e Tuberculose (4071 imagens) dos conjuntos de treino (9387 imagens) e teste (4024 imagens).

Tabela 5.2 – Distribuição das classes para a 2^a etapa.

Implementação	Tipo	Normal	Pneumonia	Tuberculose
1^a - CNN	Treinamento	3562 (37.9%)	2970 (31.6%)	2855 (30.4%)
	Teste	1513 (37.6%)	1295 (32.2%)	1216 (30.2%)
2^a - CNN	Treinamento	3549 (37.8%)	3004 (32%)	2834 (30.2%)
	Teste	1526 (37.9%)	1261 (31.3%)	1237 (30.7%)
1^a - EKAN	Treinamento	3525 (37.6%)	3004 (32%)	2858 (30.4%)
	Teste	1550 (38.5%)	1261 (31.3%)	1213 (30.1%)
2^a - EKAN	Treinamento	3562 (37.9%)	2976 (31.7%)	2849 (30.4%)
	Teste	1513 (37.6%)	1289 (32%)	1222 (30.4%)
1^a - FKAN	Treinamento	3576 (38.1%)	2958 (31.5%)	2853 (30.4%)
	Teste	1499 (37.3%)	1307 (32.5%)	1218 (30.3%)
2^a - FKAN	Treinamento	3567 (38%)	2979 (31.7%)	2841 (30.3%)
	Teste	1508 (37.5%)	1286 (32%)	1230 (30.6%)

Fonte: o autor.

A redução das implementações para cada modelo ocorreu porque somente os parâmetros das implementações da primeira etapa que tiveram os melhores resultados foram testados nesta segunda etapa, onde o grande fator foi a realização de experimentos com o banco de dados completo.

Para a terceira etapa, uma implementação foi realizada utilizando os modelos CNN, *Efficient* KAN e *Fast* KAN, com o banco de dados completo e a técnica de validação cruzada *KFold* definido para criação de três subconjuntos aleatórios. A Tabela 5.3 apresenta as descrições do banco de dados utilizado nas aplicações dos três modelos, onde nos três subconjuntos de treinamento e teste, as classes foram divididas como, Normal (5075 imagens), Pneumonia (4265 imagens) e Tuberculose (4071 imagens).

Tabela 5.3 – Distribuição das classes para a 3^a etapa.

Fold	Tipo	Normal	Pneumonia	Tuberculose
1^o	Treinamento	3362 (37.6%)	2865 (32%)	2713 (30.3%)
	Teste	1713 (38.3%)	1400 (31.3%)	1358 (30.4%)
2^o	Treinamento	3368 (37.7%)	2819 (31.5%)	2754 (30.8%)
	Teste	1707 (38.2%)	1446 (32.3%)	1317 (29.5%)
3^o	Treinamento	3420 (38.3%)	2846 (31.8%)	2675 (29.9%)
	Teste	1655 (37%)	1419 (31.7%)	1396 (31.2%)

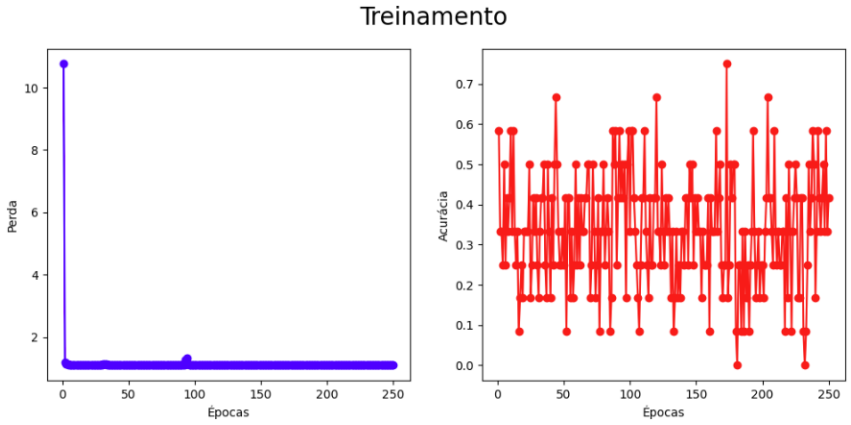
Fonte: o autor.

A apresentação destas tabelas se justifica nas definições das porcentagens de cada classe dos conjuntos de treinamento e teste, visto que, a proporcionalidade destas classes devem ser garantidas para evitar vieses que podem ser detectados pelo classificador, ou seja, uma classe específica que apresente uma pequena quantidade de dados pode ter a sua detecção prejudicada.

5.1 Resultados das implementações da 1ª etapa

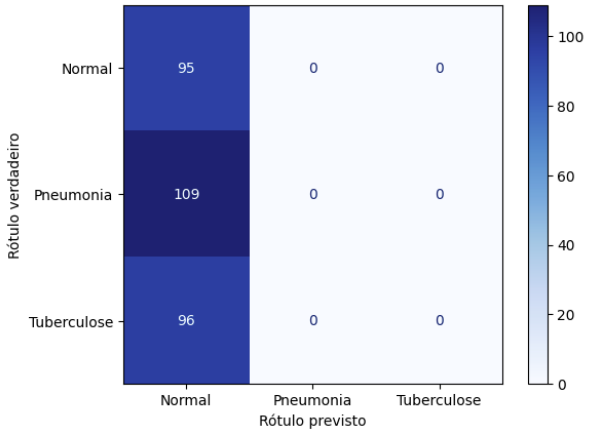
Iniciando com a primeira implementação do modelo CNN, a Figura 5.1 apresenta o desempenho do treinamento em relação à perda, mostrado pelo gráfico da esquerda, e da acurácia, mostrado pelo gráfico da direita. Para os dados de teste, a Figura 5.2 ilustra a matriz de confusão, enquanto a Tabela 5.4 apresenta as porcentagens das métricas de avaliação Acurácia, Precisão, *Recall* e pontuação *F1*.

Figura 5.1 – Treinamento da 1ª aplicação do modelo CNN.



Fonte: o autor.

Figura 5.2 – Matriz de confusão da 1ª aplicação do modelo CNN.



Fonte: o autor.

Tabela 5.4 – Métricas de avaliação dos testes da 1ª aplicação do modelo CNN.

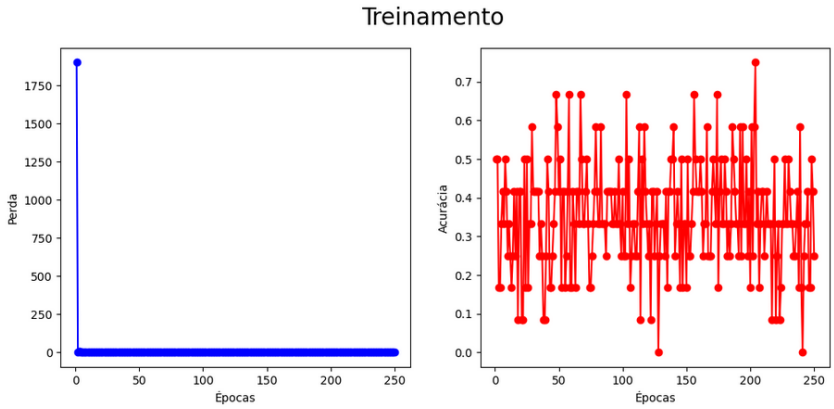
Métricas	**Valor**
Acurácia	31.67%
Precisão	10.03%
Recall	31.67%
F1	15.23%

Fonte: o autor.

Na Figura 5.1, o gráfico da esquerda, relativo ao comportamento da perda, foi finalizado com o valor 1.1. Por outro lado, no gráfico da direita, o comportamento da acurácia apresentou uma grande variação, indo de 0% até 75%, finalizando com 42%, sendo este um resultado que representa um treinamento ruim, o que leva a detecções prejudiciais de novos dados, ilustrado pela matriz de confusão da Figura 5.2, onde somente as imagens da classe Normal foram identificadas de maneira correta na sua totalidade. Por outro lado, todas as imagens das classes Pneumonia e Tuberculose foram classificadas de maneira incorreta. Isto pode ser justificado devido desempenho apresentado no treinamento, visto que, mesmo que o valor da perda seja pequeno, o que a primeira vista pode ser considerado um sucesso, o valor da acurácia também apresentou um valor de porcentagem pequeno, configurando-se como uma situação de *Underfitting*, onde o modelo não conseguiu identificar os padrões necessários para identificar todas as classes de maneira correta. Somente a classe Normal teve os seus padrões assimilados pelo modelo, sendo esta a única classe a ser identificada de maneira correta, o que não pode ser observado nas classes restantes. Estas observações têm respaldo na Tabela 5.4, que apresenta as porcentagens das métricas de avaliação Acurácia, Precisão, *Recall* e pontuação *F1* para os dados de teste.

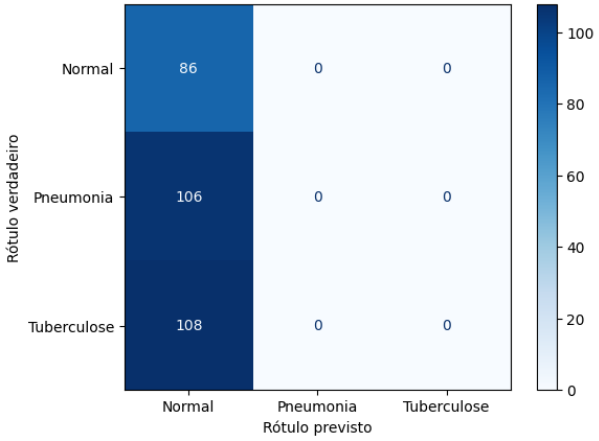
Para a segunda implementação do modelo CNN, a Figura 5.3 apresenta o comportamento da perda, no gráfico da esquerda, e da acurácia, no gráfico da direita. Para os dados de teste, a Figura 5.4 ilustra a matriz de confusão, enquanto a Tabela 5.5 apresenta as porcentagens das métricas de avaliação Acurácia, Precisão, *Recall* e pontuação *F1*.

Figura 5.3 – Treinamento da 2ª aplicação do modelo CNN.



Fonte: o autor.

Figura 5.4 – Matriz de confusão da 2ª aplicação do modelo CNN.



Fonte: o autor.

Tabela 5.5 – Métricas de avaliação dos testes da 2ª aplicação do modelo CNN.

Métricas	**Valor**
Acurácia	28.67%
Precisão	8.22%
Recall	28.67%
F1	12.77%

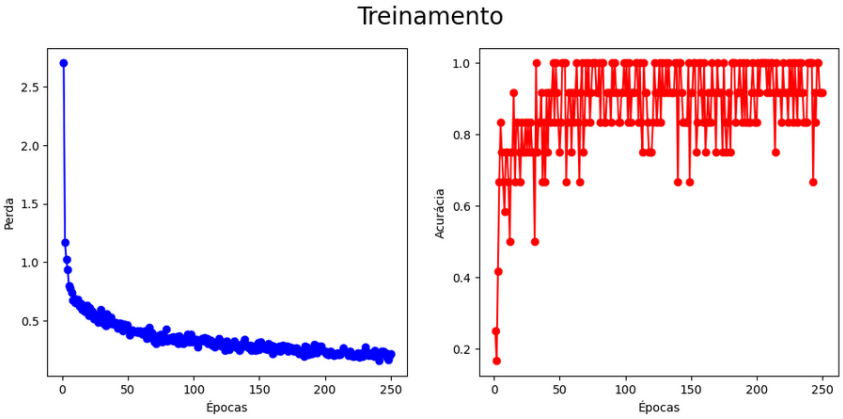
Fonte: o autor.

Como mostrado pela Figura 5.3, o comportamento apresentado foi semelhante ao demonstrado na primeira aplicação do modelo CNN, ou seja, situação de *Underfitting*. Somente com as modificações realizadas em relação à taxa de aprendizagem, ou seja, atualização dinâmica utilizando a técnica *StepLR* e valor inicial igual a 0.1, finalizou com o valor de perda 1.1, semelhante ao que foi apresentado na aplicação anterior. Já o valor da acurácia apresentou grandes variações em seu comportamento, finalizando em 25%. A Figura 5.4 ilustra a matriz de confusão desta segunda aplicação do modelo CNN. De maneira semelhante à aplicação anterior, somente a classe Normal foi identificada de maneira correta, demonstrando que, mesmo com a taxa de aprendizagem diminuindo a cada 50 épocas, o valor inicial deste parâmetro teve grande impacto, sendo necessário mais testes com este parâmetro iniciando com valores diferentes. De maneira semelhante ao apresentado na primeira implementação, a Tabela 5.5 apresenta os resultados dos testes pelas métricas de avaliação Acurácia, Precisão, *Recall* e pontuação *F1*.

Para a terceira implementação do modelo CNN, a Figura 5.5 apresenta o comportamento da perda, mostrado no gráfico da esquerda, e da acurácia, mostrado do gráfico da direita, onde a diferença para as implementações anteriores está na maneira em que a taxa de aprendizagem foi implementada, ou seja, sem atualização dinâmica e com valor inicial igual a 0.001. Para os dados de teste, a Figura 5.6 ilustra a matriz de confusão, enquanto a Tabela 5.6 apresenta as porcentagens das métricas de avaliação Acurácia, Precisão, *Recall* e

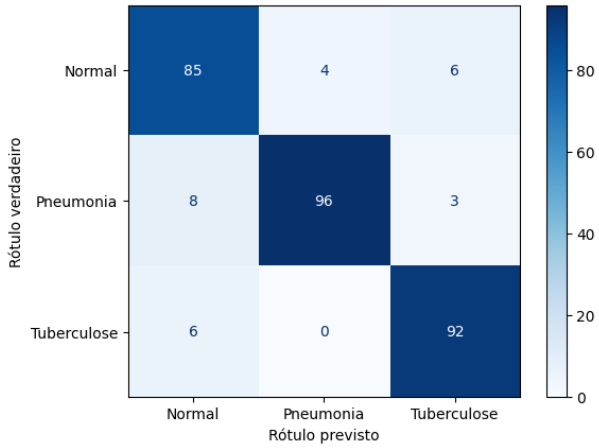
pontuação *F1*.

Figura 5.5 – Treinamento da 3^a aplicação do modelo CNN.



Fonte: o autor.

Figura 5.6 – Matriz de confusão da 3^a aplicação do modelo CNN.



Fonte: o autor.

Tabela 5.6 – Métricas de avaliação dos testes da 3^a aplicação do modelo CNN.

Métricas	**Valor**
Acurácia	91%
Precisão	91.18%
Recall	91%
F1	91.04%

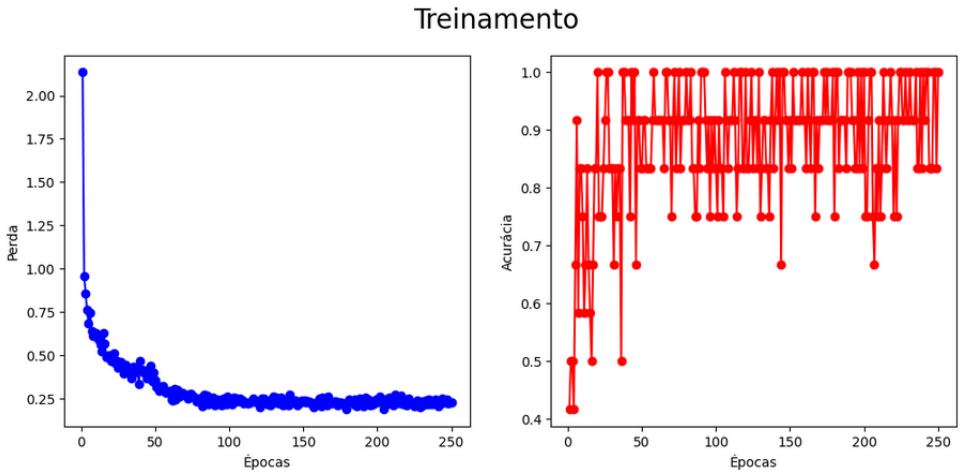
Fonte: o autor.

Como demonstrado pela Figura 5.5, o desempenho da perda apresentou uma queda gradual em relação às aplicações anteriores, finalizando em 0.21. Por outro lado, para o desempenho da acurácia, as grandes variações que ocorreram nas aplicações anteriores não aconteceram nesta implementação, finalizando com 92%. Tanto os valores apresentados na Tabela 5.6, sendo esta a primeira aplicação em que todas as porcentagens estão acima de

90%, quanto às classificações feitas de maneira correta, apresentadas na diagonal principal da matriz de confusão, demonstrada pela Figura 5.6, confirmam que a escolha de um menor valor para a taxa de aprendizagem melhorou significativamente os resultados.

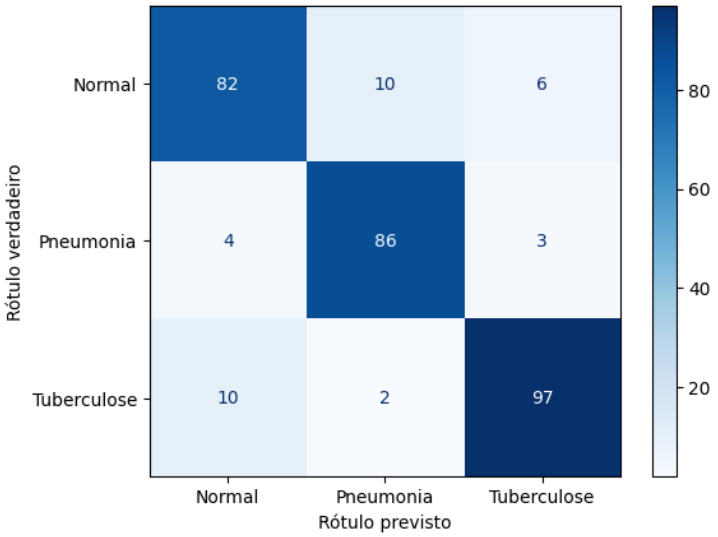
Para a quarta implementação do modelo CNN, a Figura 5.7 apresenta o comportamento da perda, mostrado pelo gráfico da esquerda, e da acurácia, mostrado pelo gráfico da direita. Para esta aplicação, manteve-se o valor inicial da taxa de aprendizagem em 0.001 utilizado na aplicação anterior, mas com atualização dinâmica por meio da técnica *StepLR*. Para os dados de teste, a Figura 5.8 apresenta a matriz de confusão, enquanto a Tabela 5.7 apresenta os resultados por meio das métricas de avaliação Acurácia, Precisão, *Recall* e pontuação *F1*.

Figura 5.7 – Treinamento da 4ª aplicação do modelo CNN.



Fonte: o autor.

Figura 5.8 – Matriz de confusão da 4ª aplicação do modelo CNN.



Fonte: o autor.

Tabela 5.7 – Métricas de avaliação dos testes da 4^a aplicação do modelo CNN.

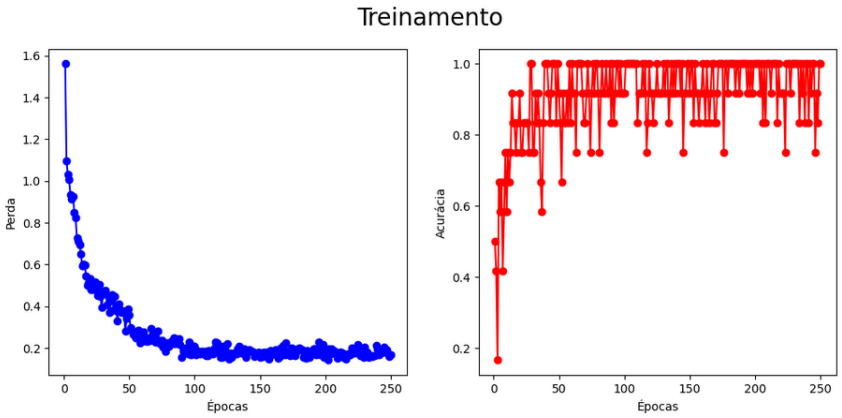
Métricas	**Valor**
Acurácia	88.33%
Precisão	88.36%
Recall	88.33%
F1	88.32%

Fonte: o autor.

Como evidenciado pela Figura 5.7, a modificação realizada para atualização dinâmica da taxa de aprendizagem não alterou de maneira significativa o desempenho do treinamento, finalizando com perda igual a 0.23 e acurácia em 100%, onde, semelhanças com o comportamento da aplicação anterior podem ser observadas, principalmente em relação ao gráfico da perda. Mesmo que os resultados apresentados pela Tabela 5.7 tenham sido inferiores aos demonstrados pelos resultados da implementação anterior, ainda podem ser considerados bons resultados, visto que, se as casas decimais forem desconsideradas, a diferença é de 2%. Também é possível observar na matriz de confusão, apresentada pela Figura 5.8, valores elevados na diagonal principal, demonstrando a eficiência deste modelo em relação aos parâmetros escolhidos.

Para a quinta implementação do modelo CNN, a Figura 5.9 apresenta o desempenho do treinamento em relação ao comportamento da perda, mostrado pelo gráfico da esquerda, e da acurácia, mostrado pelo gráfico da direita. Em relação aos dados de teste, a Figura 5.10 apresenta a matriz de confusão, enquanto a Tabela 5.8 apresenta os resultados por meio das métricas de avaliação Acurácia, Precisão, *Recall* e pontuação *F1*.

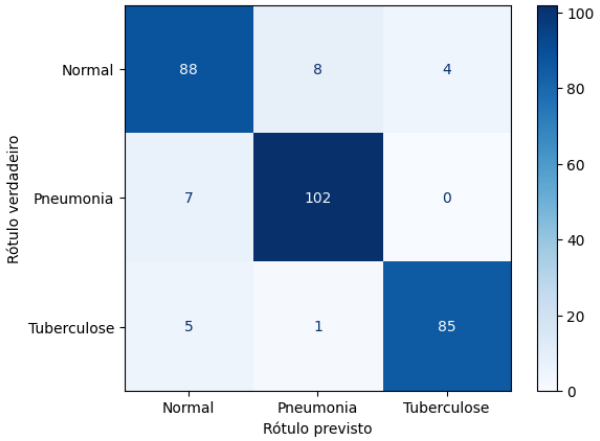
Figura 5.9 – Treinamento da 5^a aplicação do modelo CNN.



Fonte: o autor.

Devido às modificações realizadas na arquitetura, foi possível chegar em 0.17 no valor de perda, o menor valor encontrado, e 100% de acurácia. Sendo esta a implementação que obteve os melhores resultados, demonstrado pela Tabela 5.8, para a segunda etapa, o modelo CNN será utilizado com as mesmas configurações da quinta implementação, onde

Figura 5.10 – Matriz de confusão da 5ª aplicação do modelo CNN.



Fonte: o autor.

Tabela 5.8 – Métricas de avaliação dos testes da 5ª aplicação do modelo CNN.

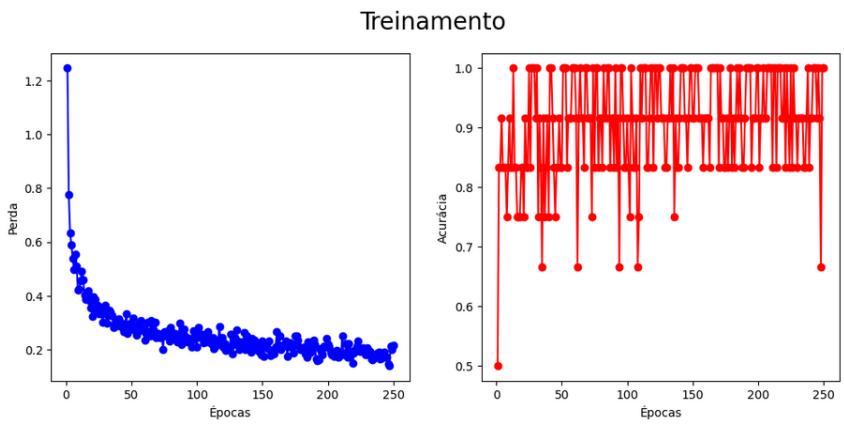
Métricas	**Valor**
Acurácia	91.67%
Precisão	91.69%
Recall	91.67%
F1	91.67%

Fonte: o autor.

foram utilizadas a arquitetura do Código 4.13, 250 épocas, taxa de aprendizagem em 0.001 (Com e Sem *StepLR*), otimizador *Adam* e função de perda *CrossEntropyLoss*.

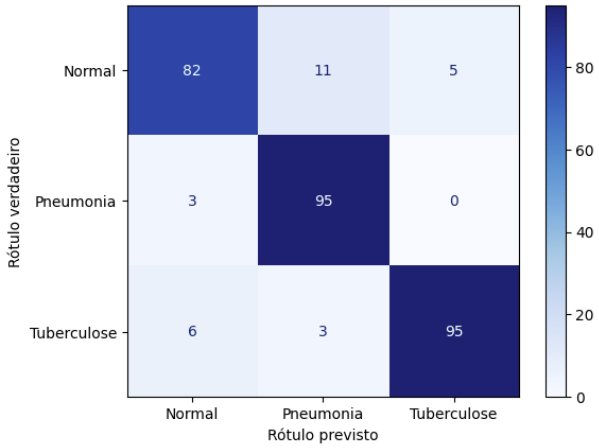
Para a primeira implementação do modelo *Efficient* KAN, a Figura 5.11 apresenta o desempenho do treinamento em relação à perda, mostrado pelo gráfico da esquerda, e da acurácia, mostrado pelo gráfico da direita. Em relação aos dados de teste, a Figura 5.12 apresenta a matriz de confusão, enquanto a Tabela 5.9 apresenta os resultados por meio das métricas de avaliação Acurácia, Precisão, *Recall* e pontuação *F1*.

Figura 5.11 – Treinamento da 1ª aplicação do modelo *Efficient* KAN.



Fonte: o autor.

Figura 5.12 – Matriz de confusão da 1^a aplicação do modelo *Efficient* KAN.



Fonte: o autor.

Tabela 5.9 – Métricas de avaliação dos testes da 1^a aplicação do modelo *Efficient* KAN.

Métricas	**Valor**
Acurácia	90.67%
Precisão	90.84%
Recall	90.67%
F1	90.62%

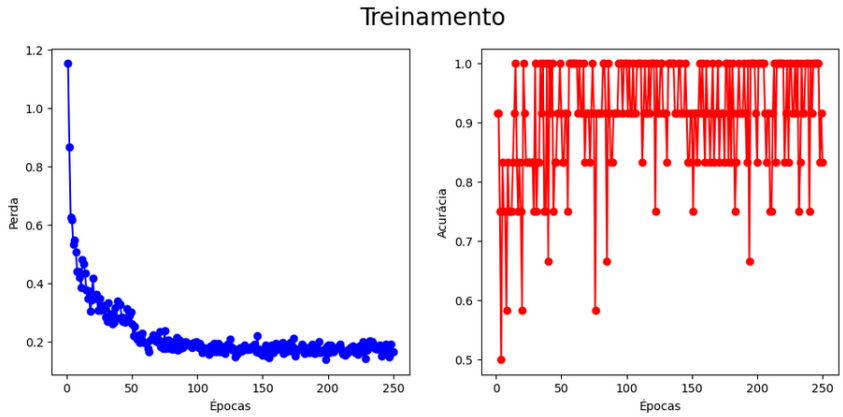
Fonte: o autor.

O desempenho apresentado pela Figura 5.11, foi semelhante ao demonstrado pelas duas últimas implementações do modelo CNN, onde este alcançou 0.22 como valor de perda e 100% de acurácia. Diante dos resultados estatísticos apresentados pela Tabela 5.9, o modelo *Efficient* KAN contendo 4 camadas, sendo elas, 1 de entrada, 2 ocultas e 1 de saída, realizou a classificação de maneira semelhante às implementações utilizando o modelo CNN com melhores resultados, sendo elas, a 3^a aplicação, com arquitetura contendo 3 camadas de convolução, 3 camadas de normalização, 3 camadas de redimensionamento, 4 camadas lineares, 3 funções para desativação de neurônios e 1 função de conversão de dados bidimensionais para unidimensionais, e a 5^a aplicação, contendo 4 camadas convolucionais, 4 camadas de normalização, 3 camadas de redimensionamento, 5 camadas lineares, 4 funções para desativação de neurônios e 1 função de conversão de dados bidimensionais para unidimensionais. A matriz de confusão ilustrada pela Figura 5.12, apresenta valores elevados para as três classes da diagonal principal, não somente evidenciando o que foi relatado anteriormente, como também demonstrando o desempenho do modelo *Efficient* KAN para classificação de imagens utilizando um número reduzido de camadas.

Para a segunda implementação do modelo *Efficient* KAN, a Figura 5.13 apresenta o desempenho da perda, mostrado pelo gráfico da esquerda, e da acurácia, mostrado pelo gráfico da direita. Para esta aplicação, manteve-se as camadas da arquitetura do modelo anterior, bem com os parâmetros de treinamento, mas com o uso da técnica *StepLR* para

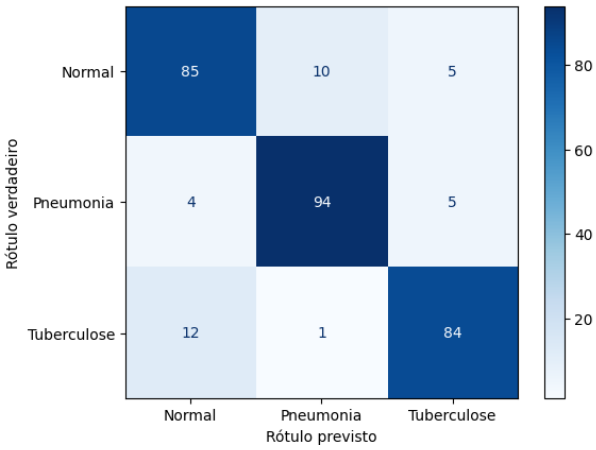
atualizar a taxa de aprendizagem. Com relação aos dados de teste, a Figura 5.14 apresenta a matriz de confusão, enquanto a Tabela 5.10 apresenta os resultados por meio das métricas de avaliação Acurácia, Precisão, *Recall* e pontuação *F1*.

Figura 5.13 – Treinamento da 2ª aplicação do modelo *Efficient* KAN.



Fonte: o autor.

Figura 5.14 – Matriz de confusão da 2ª aplicação do modelo *Efficient* KAN.



Fonte: o autor.

Tabela 5.10 – Métricas de avaliação dos testes da 2ª aplicação do modelo *Efficient* KAN.

Métricas	**Valor**
Acurácia	87.67%
Precisão	87.68%
Recall	87.67%
F1	87.66%

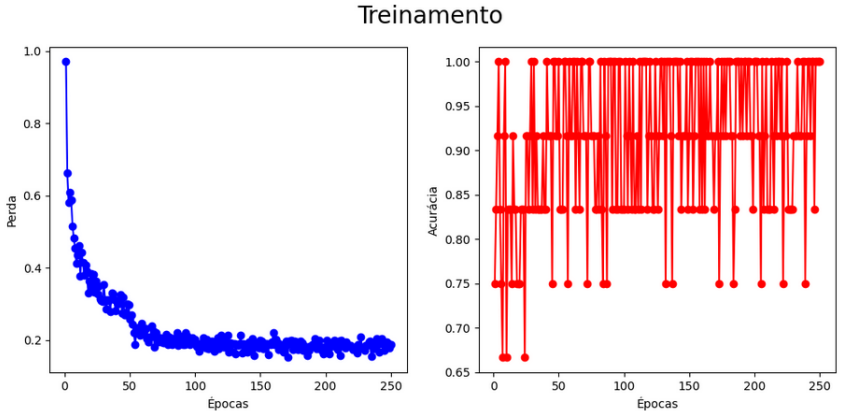
Fonte: o autor.

O comportamento do treinamento ilustrado pela Figura 5.13, chegou em 0.17 no valor da perda com 83% de acurácia. Portanto, comparando estes valores com aqueles apresentados na primeira implementação para o modelo *Efficient* KAN, pode-se entender que ocorreu

uma pequena queda em relação ao aprendizado, visto que a única alteração realizada foi na maneira em que é realizado a atualização da taxa de aprendizagem, indo de estático para dinâmico. Esta queda no desempenho do treinamento refletiu no aumento de classificações erradas na inferência nos dados de teste, podendo ser observado pela matriz de confusão, mostrado pela Figura 5.14, e pelas métricas de avaliação na Tabela 5.10.

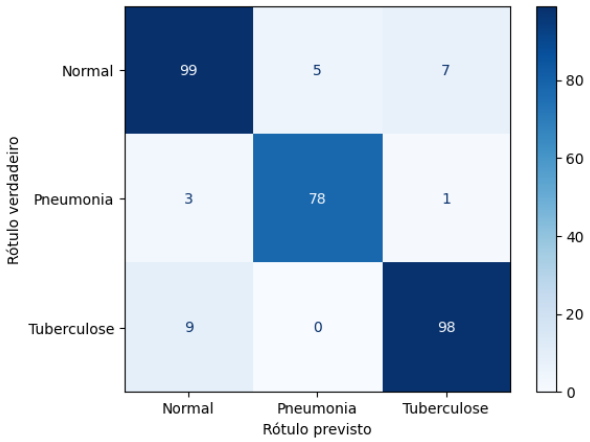
Para a terceira implementação do modelo *Efficient* KAN, com arquitetura contendo cinco camadas, o comportamento ilustrado pela Figura 5.15 apresentou o melhor desempenho em comparação com as duas aplicações utilizando o modelo *Efficient* KAN já apresentadas, onde a perda, ilustrado pelo gráfico da esquerda, chegou em 0.19, e a acurácia, mostrado pelo gráfico da direita, chegou em 100%. Para os dados de teste, a Figura 5.16 apresenta a matriz de confusão e a Tabela 5.11 apresenta as métricas de avaliação Acurácia, Precisão, *Recall* e pontuação *F1*.

Figura 5.15 – Treinamento da 3^a aplicação do modelo *Efficient* KAN.



Fonte: o autor.

Figura 5.16 – Matriz de confusão da 3^a aplicação do modelo *Efficient* KAN.



Fonte: o autor.

Tabela 5.11 – Métricas de avaliação dos testes da 3ª aplicação do modelo *Efficient* KAN.

Métricas	**Valor**
Acurácia	91.67%
Precisão	91.66%
Recall	91.67%
F1	91.66%

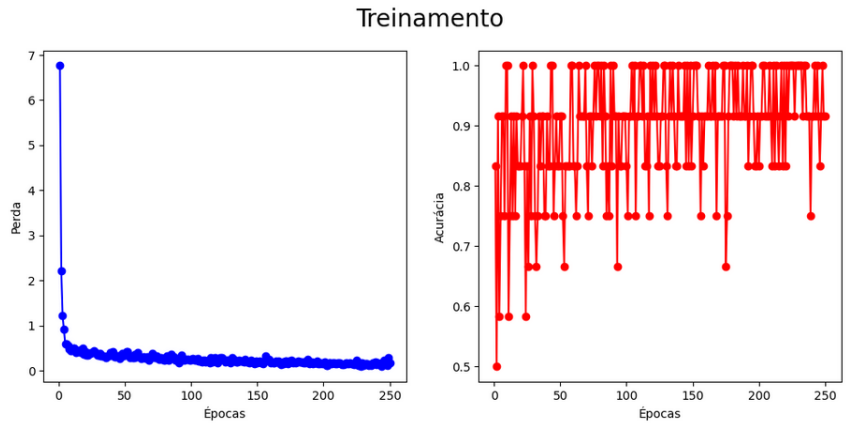
Fonte: o autor.

Esta aplicação não somente apresentou o melhor comportamento no treinamento para os modelos *Efficient* KAN nesta primeira etapa, como também alcançou um dos maiores resultados em relação às inferências realizadas com os dados de teste.

Como demonstrado pelos resultados das três aplicações utilizando o modelo *Efficient* KAN, arquitetura baseada no Teorema de Representação de Kolmogorov-Arnold, sua eficiência é semelhante ao que foi apresentado pelos resultados utilizando o modelo CNN, onde mais teste na 2ª etapa são necessários para verificar se este equilíbrio é mantido com o banco de dados completo contendo 13411 imagens.

Para a primeira aplicação do modelo *Fast* KAN, a Figura 5.17 ilustra o comportamento da perda, mostrado pelo gráfico da esquerda, e da acurácia, mostrado pelo gráfico da direita, onde os parâmetros e a arquitetura utilizadas na primeira implementação do modelo *Efficient* KAN foram replicados para esta aplicação. Em relação aos testes, a Figura 5.18 ilustra a matriz de confusão e a Tabela 5.12 apresenta as porcentagens das métricas de avaliação Acurácia, Precisão, *Recall* e pontuação *F1*.

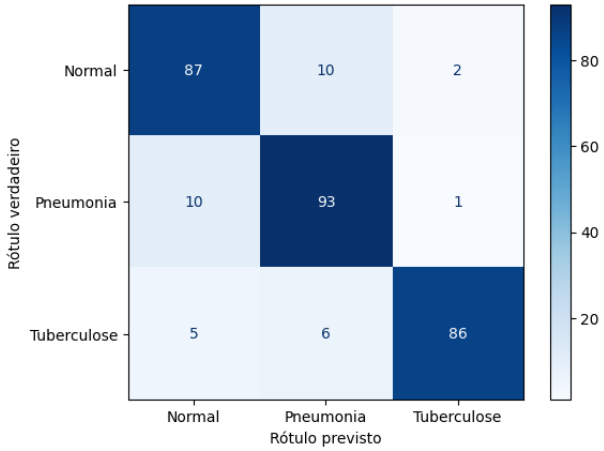
Figura 5.17 – Treinamento da 1ª aplicação do modelo *Fast* KAN.



Fonte: o autor.

O comportamento apresentado pela Figura 5.17 alcançou 0.17 como valor de perda e 92% de acurácia. Os resultados para os dados de teste foram inferiores ao apresentado pela primeira implementação do modelo *Efficient* KAN, mas ainda aceitáveis, visto que a diferença é de 2% entre as porcentagens das métricas de avaliação.

Figura 5.18 – Matriz de confusão da 1^a aplicação do modelo *Fast* KAN.



Fonte: o autor.

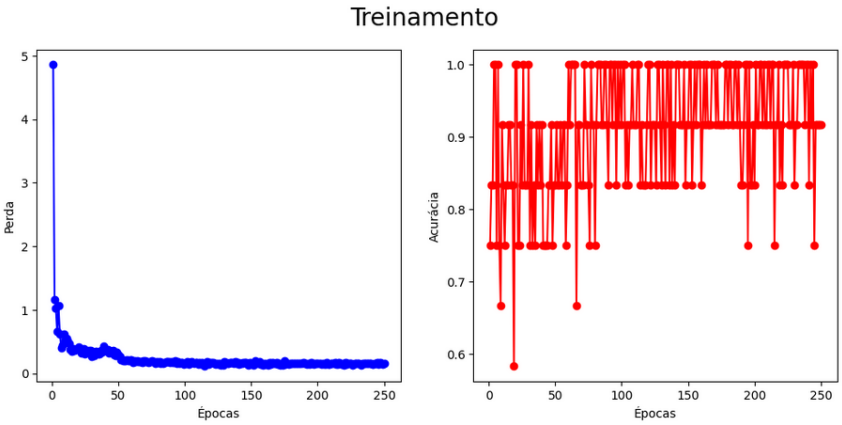
Tabela 5.12 – Métricas de avaliação dos testes da 1^a aplicação do modelo *Fast* KAN.

Métricas	**Valor**
Acurácia	88.67%
Precisão	88.97%
Recall	88.67%
F1	88.74%

Fonte: o autor.

Para a segunda implementação do modelo *Fast* KAN, onde é realizada a atualização dinâmica de taxa de aprendizagem, a Figura 5.19 ilustra o comportamento da perda, no gráfico da esquerda, e da acurácia, no gráfico da direita. Para a validação do modelo, a Figura 5.20 ilustra a matriz de confusão, enquanto a Tabela 5.13 apresenta os resultados por meio das métricas de avaliação Acurácia, Precisão, *Recall* e pontuação *F1*.

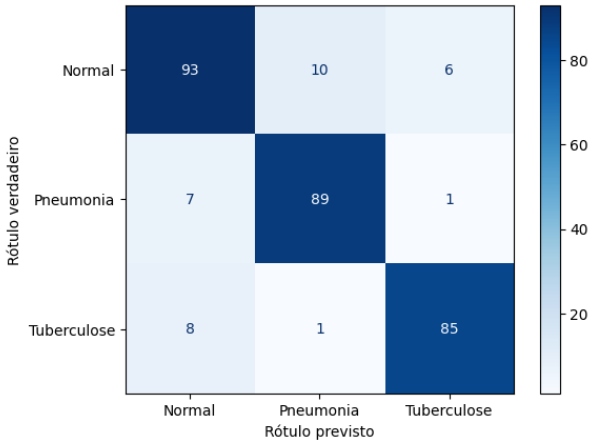
Figura 5.19 – Treinamento da 2^a aplicação do modelo *Fast* KAN.



Fonte: o autor.

O comportamento apresentado pela Figura 5.19, alcançou 0.15 como valor de perda e 92% de acurácia. A Tabela 5.13 demonstra as porcentagens das métricas de avaliação, en-

Figura 5.20 – Matriz de confusão da 2ª aplicação do modelo *Fast* KAN.



Fonte: o autor.

Tabela 5.13 – Métricas de avaliação dos testes da 2ª aplicação do modelo *Fast* KAN.

Métricas	**Valor**
Acurácia	89%
Precisão	89.01%
Recall	89%
F1	89%

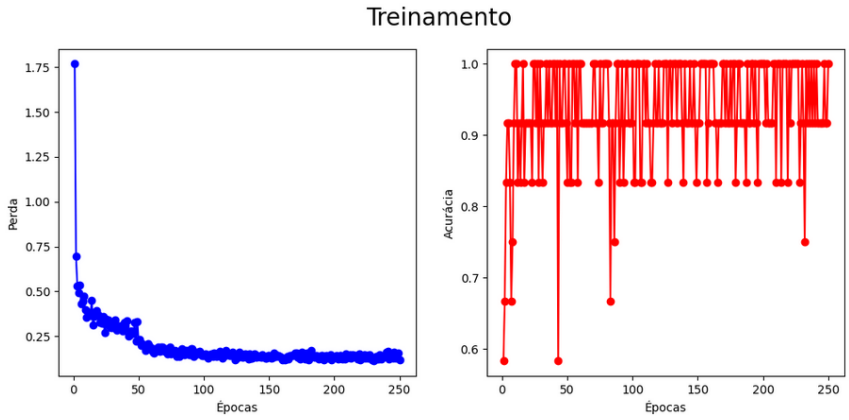
Fonte: o autor.

quanto a Figura 5.20 apresenta a matriz de confusão para os testes.

Como demonstrado pelas métricas de avaliação, uma melhora de 1% foi apresentada no desempenho de detecção das classes com a técnica *StepLR*. Tal fato não foi observado nos resultados dos modelos CNN e *Efficient* KAN.

Para a terceira aplicação do modelo *Fast* KAN, onde foi utilizado a arquitetura com 5 camadas e a técnica *StepLR*, a Figura 5.21 ilustra o comportamento da perda, no gráfico da esquerda, e da acurácia, no gráfico da direita.

Figura 5.21 – Treinamento da 3ª aplicação do modelo *Fast* KAN.



Fonte: o autor.

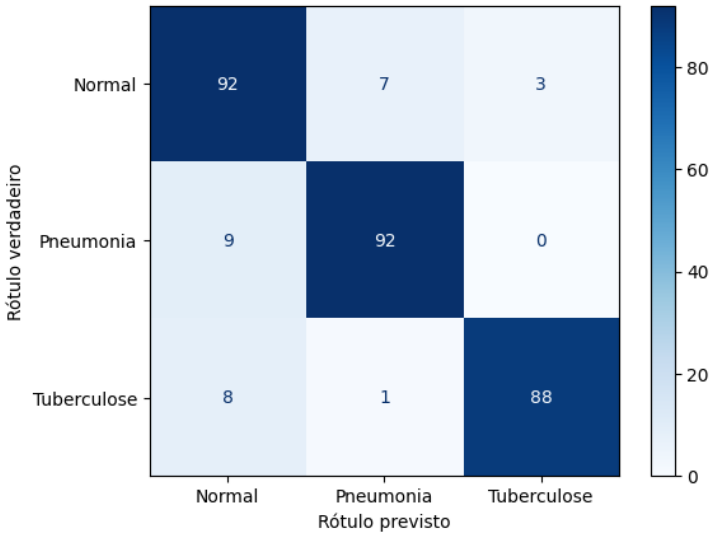
O comportamento apresentado pela Figura 5.21, alcançou 0.12 como valor de perda, sendo este o menor valor encontrado nesta primeira etapa, com 100% de acurácia. A Tabela 5.14 demonstra os valores das métricas de avaliação Acurácia, Precisão, *Recall* e pontuação *F1*, enquanto a Figura 5.22 apresenta a matriz de confusão.

Tabela 5.14 – Métricas de avaliação dos testes da 3^a aplicação do modelo *Fast* KAN.

Métricas	**Valor**
Acurácia	90.67%
Precisão	90.94%
Recall	90.67%
F1	90.74%

Fonte: o autor.

Figura 5.22 – Matriz de confusão da 3^a aplicação do modelo *Fast* KAN.



Fonte: o autor.

Como observado pela Tabela 5.14, esta terceira aplicação para o modelo *Fast* KAN alcançou resultados semelhantes aos maiores valores desta primeira etapa, com diferença de 1%. Portanto, para a segunda etapa de testes, esta arquitetura será utilizada para identificar as classes Normal, Pneumonia e Tuberculose no banco de dados completo contendo 13411 imagens em duas situações, sendo a primeira, sem atualização dinâmica da taxa de aprendizagem, e a segunda, com o uso da técnica *StepLR*. Estas duas situações de testes não ocorreram somente com o modelo *Fast* KAN, mas também com os modelos CNN e *Efficient* KAN. A Tabela 5.15 apresenta o tempo de execução de todas as implementações realizadas nesta etapa.

Tabela 5.15 – Tempo de execução para as implementações da 1ª etapa.

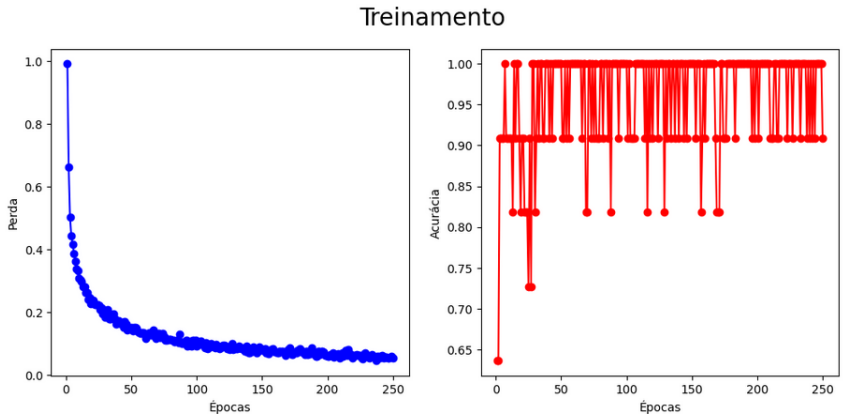
Implementações	CNN	*Efficient* KAN	*Fast* KAN
1ª	1 h 17 min	1 h 17 min	1 h 04 min
2ª	1 h 15 min	1 h 21 min	1 h 03 min
3ª	1 h 17 min	1 h 13 min	1 h 04 min
4ª	1 h 20 min		
5ª	1 h 14 min		

Fonte: o autor.

5.2 Resultados das implementações da 2ª etapa

Para a primeira implementação do modelo CNN, a Figura 5.23 apresenta o comportamento da perda, no gráfico da esquerda, e da acurácia, no gráfico da direita. A arquitetura demonstrada no Código 4.13, otimizador *Adam*, função de perda *CrossEntropyLoss*, 250 épocas e taxa de aprendizagem em 0.001, de maneira estática, foram utilizadas nesta aplicação. Para demonstrar as inferências realizadas nos dados de teste, de maneira gráfica e estatística, a Figura 5.24 ilustra a matriz de confusão e a Tabela 5.16 apresenta as métricas de avaliação Acurácia, Precisão, *Recall* e pontuação *F1*.

Figura 5.23 – Treinamento da 1ª aplicação do modelo CNN.



Fonte: o autor.

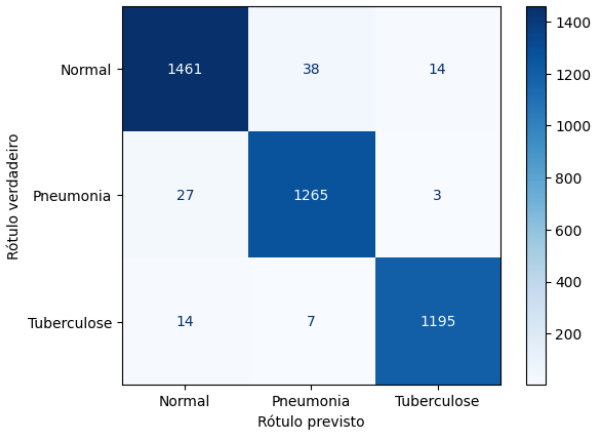
Tabela 5.16 – Métricas de avaliação dos testes da 1ª aplicação do modelo CNN.

Métricas	Valor
Acurácia	97.44%
Precisão	97.44%
Recall	97.44%
F1	97.44%

Fonte: o autor.

O comportamento apresentado pela Figura 5.23, alcançou 0.05 em valor de perda com 91% de acurácia. Devido à utilização do banco de dados completo contendo 13411 imagens,

Figura 5.24 – Matriz de confusão da 1^a aplicação do modelo CNN.

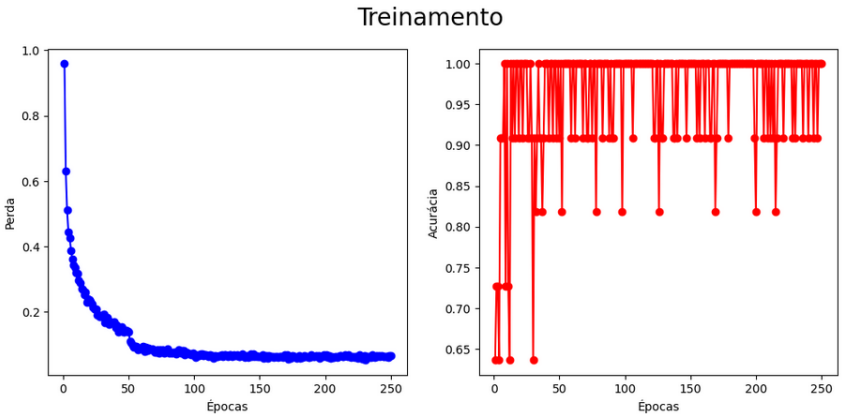


Fonte: o autor.

tanto o desempenho do treinamento, quanto às porcentagens das métricas de avaliação, chegando em 97%, demonstram que a arquitetura escolhida foi capaz de identificar de maneira correta grande parte das classes Normal, Pneumonia e Tuberculose, como pode ser observado na Figura 5.24 referente a matriz de confusão.

Para a segunda implementação do modelo CNN, a Figura 5.25 apresenta o comportamento da perda, mostrado pelo gráfico da esquerda, e da acurácia, mostrado pelo gráfico da direita. Nesta aplicação, a mesma arquitetura e parâmetros utilizados na implementação anterior foram mantidas, com exceção da atualização da taxa de aprendizagem, que passou de estática para dinâmica, com o uso da técnica *StepLR*. Para os testes, a Figura 5.26 ilustra a matriz de confusão, enquanto a Tabela 5.17 apresenta as métricas de avaliação Acurácia, Precisão, *Recall* e pontuação *F1*.

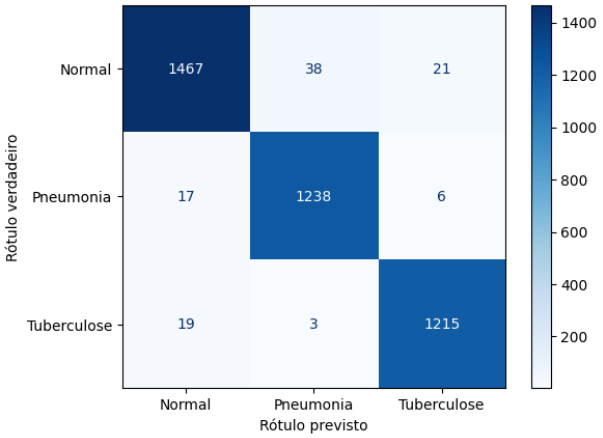
Figura 5.25 – Treinamento da 2^a aplicação do modelo CNN.



Fonte: o autor.

O desempenho apresentado no treinamento alcançou 0.06 em valor de perda com 100% de acurácia. Para a segunda etapa, a única diferença entre a primeira e a segunda implementação do modelo CNN está no uso da técnica *StepLR* para atualização dinâmica

Figura 5.26 – Matriz de confusão da 2ª aplicação do modelo CNN.



Fonte: o autor.

Tabela 5.17 – Métricas de avaliação dos testes da 2ª aplicação do modelo CNN.

Métricas	**Valor**
Acurácia	97.42%
Precisão	97.42%
Recall	97.42%
F1	97.42%

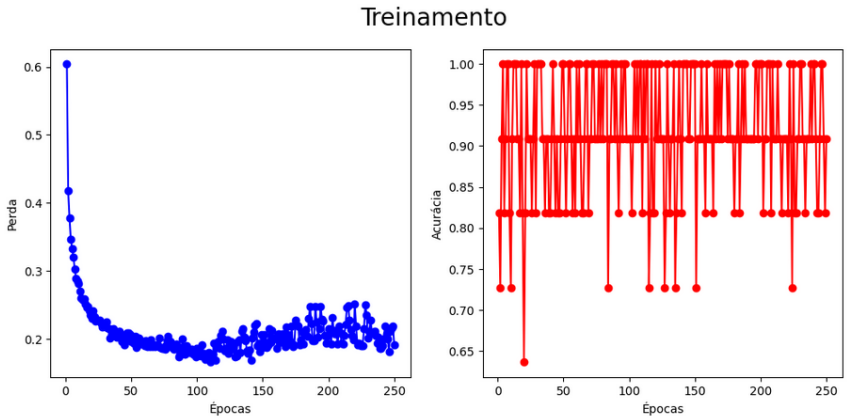
Fonte: o autor.

da taxa de aprendizagem. Visto os bons resultados encontrados nesta aplicação, bem como as semelhanças que podem ser observadas tanto no desempenho do treinamento, quanto na detecção de novas imagens entre as duas aplicações pelo modelo CNN, o uso da técnica *StepLR* não alterou de maneira significativa o aprendizado do modelo e a identificação de imagens nos dados de teste. Portanto, para a terceira etapa, a técnica *StepLR* não será utilizada, mantendo a arquitetura e os parâmetros de treinamento.

Para a primeira implementação do modelo *Efficient* KAN, a Figura 5.27 apresenta o desempenho da perda, mostrado pelo gráfico da esquerda, e da acurácia, mostrado pelo gráfico da direita. Foram utilizadas nesta aplicação a arquitetura demonstrada no Código 4.15, otimizador *Adam*, função de perda *CrossEntropyLoss*, 250 épocas e taxa de aprendizagem em 0.001, de maneira estática. Os testes realizados para este modelo, de maneira gráfica e estatística, podem ser observadas pela matriz de confusão na Figura 5.28 e pelas métricas de avaliação Acurácia, Precisão, *Recall* e pontuação *F1* na Tabela 5.18.

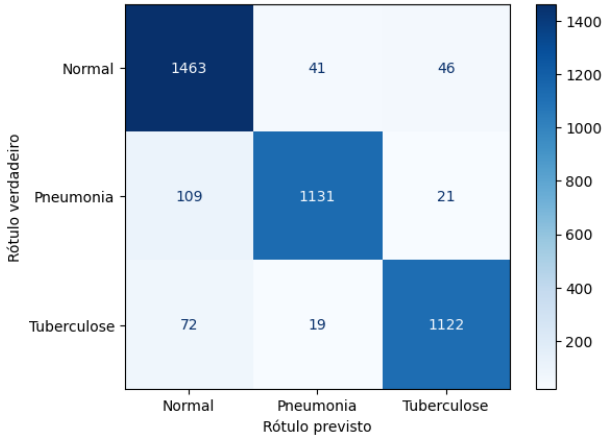
O desempenho apresentado no treinamento alcançou 0.19 em valor de perda com 91% de acurácia. Com o uso do banco de dados completo, foi possível observar uma pequena melhora nos resultados mostrados pela Tabela 5.18, mas não como ocorreu para o modelo CNN, onde foi identificada uma diferença de 1% entre esta aplicação e a terceira aplicação do modelo *Efficient* KAN na primeira etapa.

Figura 5.27 – Treinamento da 1^a aplicação do modelo *Efficient* KAN.



Fonte: o autor.

Figura 5.28 – Matriz de confusão da 1^a aplicação do modelo *Efficient* KAN.



Fonte: o autor.

Tabela 5.18 – Métricas de avaliação dos testes da 1^a aplicação do modelo *Efficient* KAN.

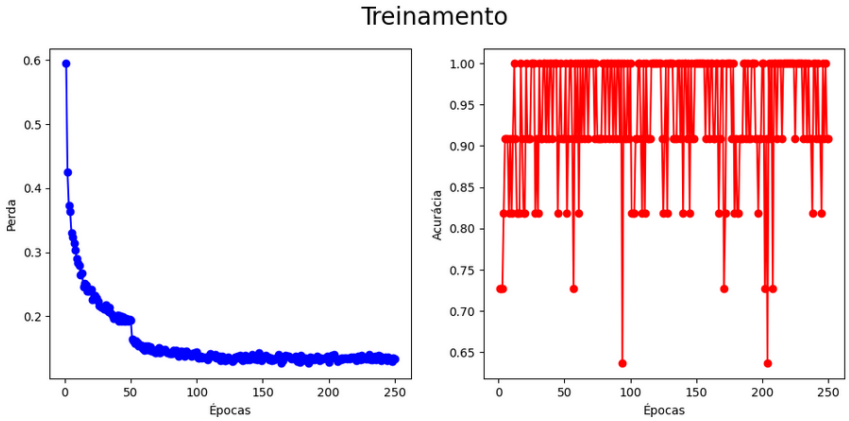
Métricas	**Valor**
Acurácia	92.35%
Precisão	92.48%
Recall	92.35%
F1	92.36%

Fonte: o autor.

Para a segunda implementação do modelo *Efficient* KAN, a Figura 5.29 apresenta o comportamento da perda, mostrado pelo gráfico da esquerda, e da acurácia, mostrado pelo gráfico da direita. Nesta aplicação, foram utilizadas a mesma arquitetura e parâmetros de treinamento da implementação anterior, mas com a diferença do uso da técnica *StepLR*. Para os testes, a Figura 5.30 ilustra a matriz de confusão e a Tabela 5.19 apresenta os resultados por meio das métricas de avaliação Acurácia, Precisão, *Recall* e pontuação *F1*.

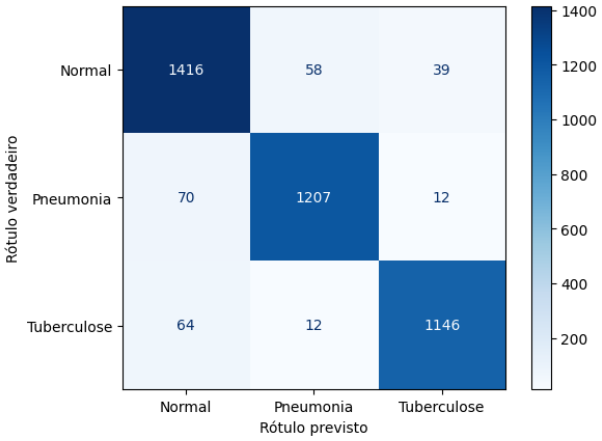
O desempenho apresentado no treinamento alcançou 0.13 em valor de perda com

Figura 5.29 – Treinamento da 2ª aplicação do modelo *Efficient* KAN.



Fonte: o autor.

Figura 5.30 – Matriz de confusão da 2ª aplicação do modelo *Efficient* KAN.



Fonte: o autor.

Tabela 5.19 – Métricas de avaliação dos testes da 2ª aplicação do modelo *Efficient* KAN.

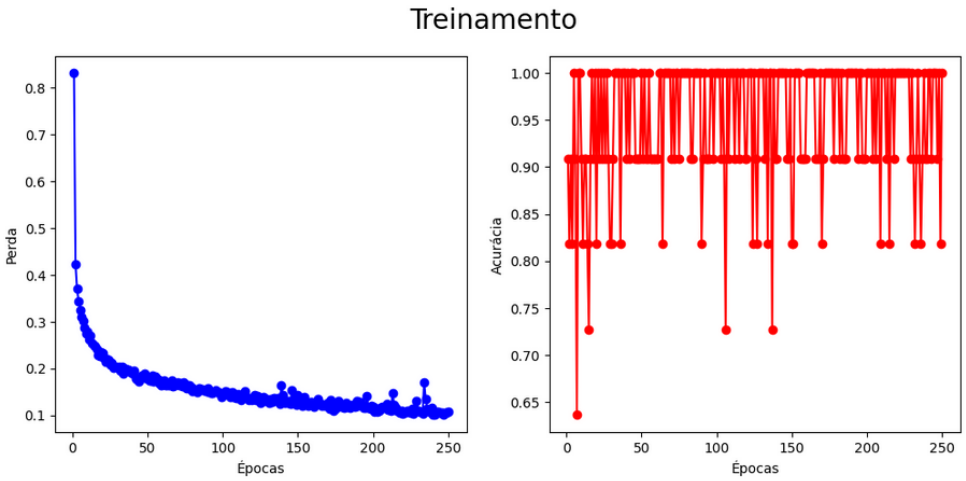
Métricas	**Valor**
Acurácia	93.66%
Precisão	93.70%
Recall	93.66%
F1	93.67%

Fonte: o autor.

91% de acurácia. Para a segunda etapa, semelhante ao ocorrido pelos testes com o modelo CNN, a única diferença entre as implementações para o algoritmo *Efficient* KAN, está no uso da técnica para atualização dinâmica *StepLR* da taxa de aprendizagem. Mesmo que resultados demonstrados pela Tabela 5.19, sejam considerados bons, somente uma diferença de 1% pode ser observado entre os resultados da primeira e segunda aplicação para o modelo *Efficient* KAN, o que denota a falta de impacto que a técnica *StepLR* teve nos resultados. Portanto, para a terceira etapa, a atualização dinâmica não será utilizada, somente os parâmetros do treinamento e a arquitetura.

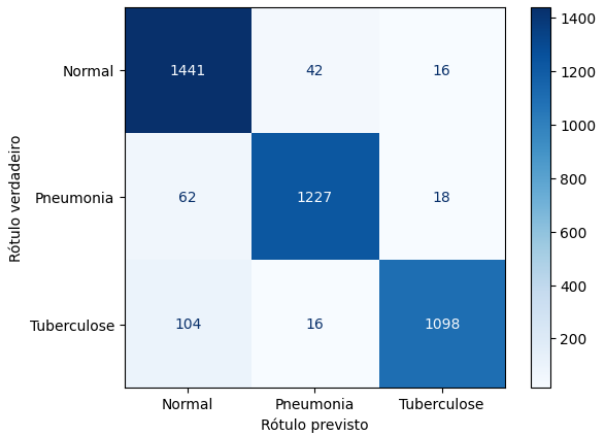
Para a primeira implementação do modelo *Fast* KAN, a Figura 5.31 apresenta o comportamento da perda, mostrado pelo gráfico da esquerda, e da acurácia, mostrado pelo gráfico da direita. Foram utilizadas para esta aplicação a arquitetura demonstrada pelo Código 4.17, otimizador *Adam*, função de perda *CrossEntropyLoss*, 250 épocas e taxa de aprendizagem com valor inicial em 0.001, de maneira estática. As inferências realizadas nas imagens no conjunto de teste podem ser observadas, de maneira gráfica, pela matriz de confusão na Figura 5.32, e de maneira estatística, pelas métricas de avaliação Acurácia, Precisão, *Recall* e pontuação *F1* na Tabela 5.20.

Figura 5.31 – Treinamento da 1^a aplicação do modelo *Fast* KAN.



Fonte: o autor.

Figura 5.32 – Matriz de confusão da 1^a aplicação do modelo *Fast* KAN.



Fonte: o autor.

O desempenho apresentado no treinamento alcançou 0.11 em valor de perda com 100% de acurácia. Com o banco de dados completo, foi possível observar uma melhora de 3% nos resultados, mostrados pela Tabela 5.20, sendo este uma melhora relativamente maior ao que foi apresentado pelo modelo *Efficient* KAN, mas menor do que o modelo CNN.

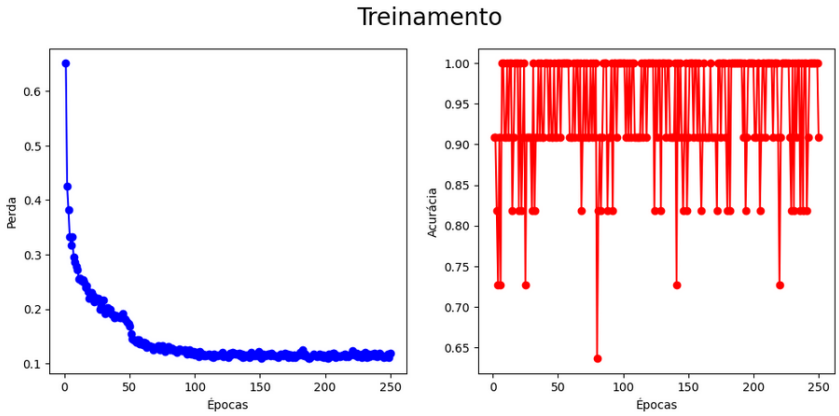
Tabela 5.20 – Métricas de avaliação dos testes da 1ª aplicação do modelo *Fast* KAN.

Métricas	**Valor**
Acurácia	93.59%
Precisão	93.78%
Recall	93.59%
F1	93.60%

Fonte: o autor.

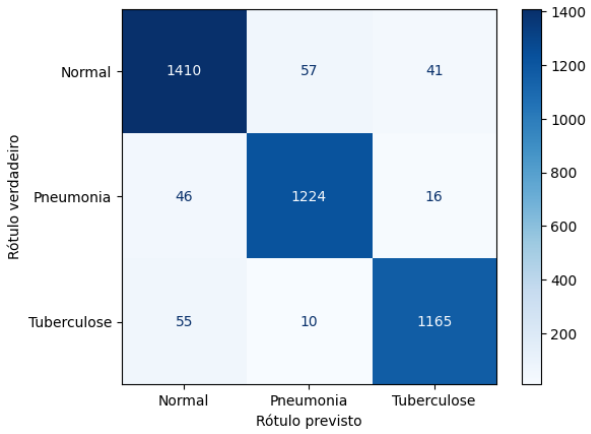
Para a segunda implementação do modelo *Fast* KAN, a Figura 5.33 apresenta o desempenho da perda, mostrado pelo gráfico da esquerda, e da acurácia, mostrado pelo gráfico da direita. Nesta aplicação, foram utilizados os mesmos parâmetros e arquitetura da implementação anterior, exceto pelo uso da técnica *StepLR* para atualização da taxa de aprendizagem. Para as classificações das imagens de teste, a Figura 5.34 ilustra a matriz de confusão, enquanto a Tabela 5.21 apresenta os resultados para as métricas de avaliação Acurácia, Precisão, *Recall* e pontuação *F1*.

Figura 5.33 – Treinamento da 2ª aplicação do modelo *Fast* KAN.



Fonte: o autor.

Figura 5.34 – Matriz de confusão da 2ª aplicação do modelo *Fast* KAN.



Fonte: o autor.

Tabela 5.21 – Métricas de avaliação dos testes da 2^a aplicação do modelo *Fast* KAN.

Métricas	**Valor**
Acurácia	94.41%
Precisão	94.41%
Recall	94.41%
F1	94.41%

Fonte: o autor.

O comportamento apresentado no treinamento, resultou em 0.12 como valor de perda e 91% de acurácia. Como demonstrado pelas métricas de avaliação, na Tabela 5.21 e pela presença de imagens classificadas de maneira correta na diagonal principal da matriz de confusão, na Figura 5.34, esta aplicação apresentou o melhor resultado para os algoritmos baseados na arquitetura KAN, onde o uso da técnica *StepLR* aumentou em 1% as porcentagens das métricas de avaliação, comparando com os resultados da primeira implementação do modelo *Fast* KAN na segunda etapa. A Tabela 5.22 apresenta o tempo de execução para cada implementação para os modelos CNN, *Efficient* KAN e *Fast* KAN.

Tabela 5.22 – Tempo de execução para as implementações da 2^a etapa.

Implementações	**CNN**	***Efficient* KAN**	***Fast* KAN**
1^a	14 h 56 min	12 h 44 min	12 h 12 min
2^a	15 h 14 min	12 h 45 min	11 h 34 min

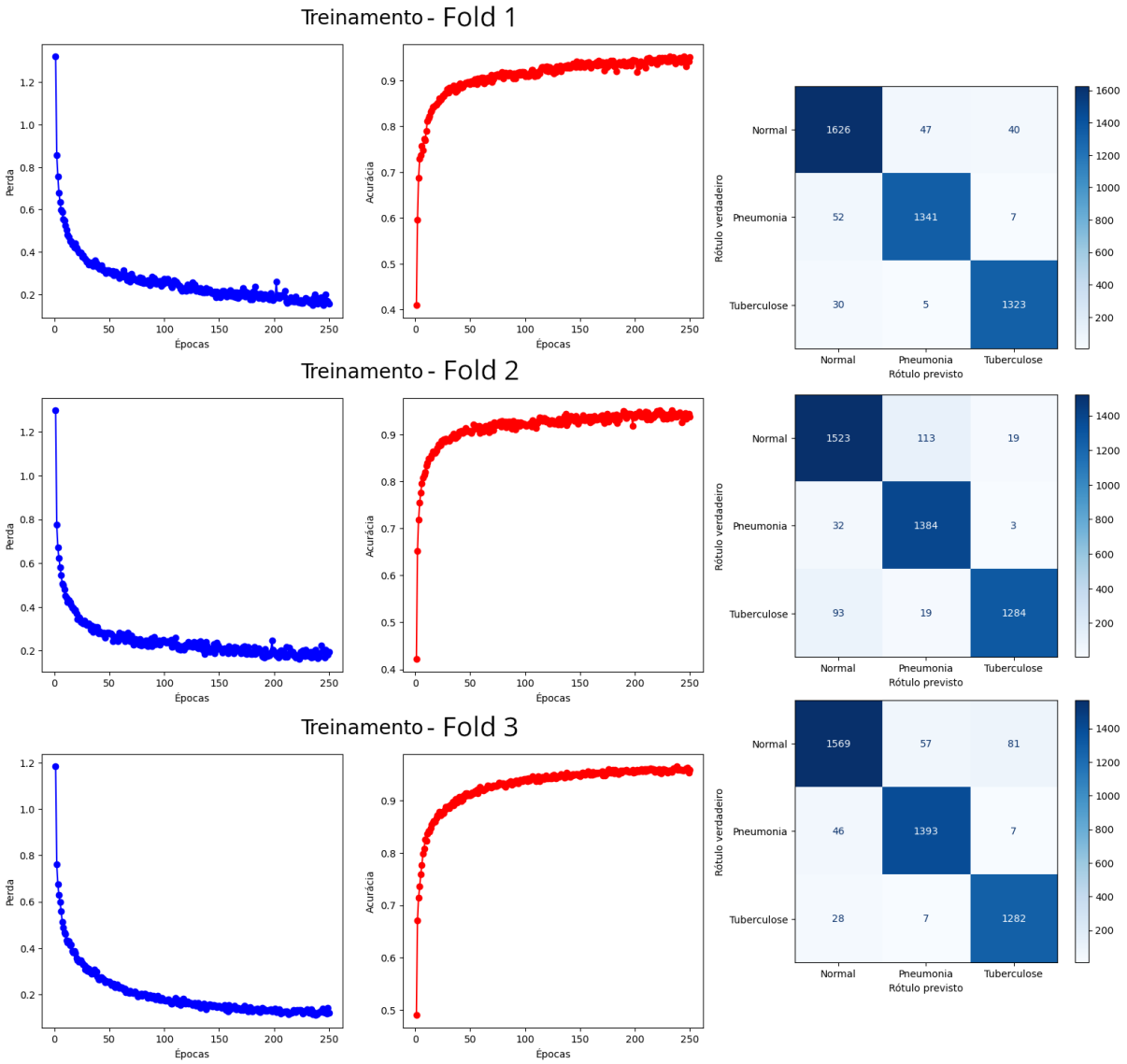
Fonte: o autor.

5.3 Resultados das implementações da 3^a etapa

Iniciando com implementação do modelo CNN, para os três *Folds*, a Figura 5.35 apresenta o desempenho do treinamento em relação à perda, mostrado pelos gráficos da esquerda, e da acurácia, mostrado pelos gráficos da direita, enquanto as matrizes de confusão referente às classificações dos dados de teste, para cada *Fold*, são apresentadas no lado direto dos gráficos da parte de treinamento. Nessa aplicação, foram utilizados os mesmos parâmetros de treinamento, sem o uso da técnica *StepLR*, e a mesma arquitetura da implementação realizada na segunda etapa. Para os dados de teste, a Tabela 5.23 apresenta as porcentagens das métricas de avaliação Acurácia, Precisão, *Recall* e pontuação *F1* para cada *Fold*, enquanto a Tabela 5.24 apresenta as médias destas métricas.

O comportamento apresentado no treinamento para o primeiro *Fold*, resultou em 0.16 como valor de perda e 95.07% de acurácia, para o segundo *Fold*, foi encontrado o valor de perda 0.12, com 95.81% de acurácia, e para o terceiro *Fold*, foi encontrado um valor de perda igual a 0.19, com 93.74% de acurácia.

Figura 5.35 – Treinamento e matriz de confusão para o modelo CNN.



Fonte: o autor.

Tabela 5.23 – Métricas de avaliação para cada *Fold* no modelo CNN

Folds	**Acurácia**	**Precisão**	***Recall***	**F1**
1	95.95%	95.95%	95.95%	95.95%
2	94.94%	94.97%	94.94%	94.93%
3	93.76%	93.90%	93.76%	93.76%

Fonte: o autor.

Como demonstrado pelas classificações corretas nas três matrizes de confusão como também pelas médias das métricas de avaliação, o modelo CNN aplicando com a técnica de validação cruzada *KFold* no banco de dados completo, alcançou, aproximadamente, 95% para Acurácia, Precisão, *Recall* e pontuação *F1*, sendo este o resultado definitivo para a arquitetura CNN.

Tabela 5.24 – Média das métricas de avaliação para o modelo CNN.

Métricas	**Média**
Acurácia	94.88%
Precisão	94.94%
Recall	94.88%
F1	94.88%

Fonte: o autor.

Para a implementação do modelo *Efficient* KAN, a Figura 5.36 apresenta o desempenho do treinamento realizado para cada *Fold* em relação à perda, mostrado pelos gráficos da esquerda, e da acurácia, mostrado pelos gráficos da direita, com suas respectivas matrizes de confusão. Nessa aplicação, a arquitetura e os parâmetros de treinamento da última implementação do modelo *Efficient* KAN na segunda etapa foram utilizados, com exceção da técnica *StepLR*. Em relação aos testes do modelo, a Tabela 5.25 apresenta as métricas de avaliação para cada *Fold*, enquanto a Tabela 5.26 demonstra as médias destas métricas.

Tabela 5.25 – Métricas de avaliação para cada *Fold* no modelo *Efficient* KAN.

Folds	**Acurácia**	**Precisão**	***Recall***	**F1**
1	92.31%	92.41%	92.31%	92.30%
2	90.87%	91.13%	90.87%	90.91%
3	93.22%	93.26%	93.22%	93.20%

Fonte: o autor.

Tabela 5.26 – Média das métricas de avaliação para o modelo *Efficient* KAN.

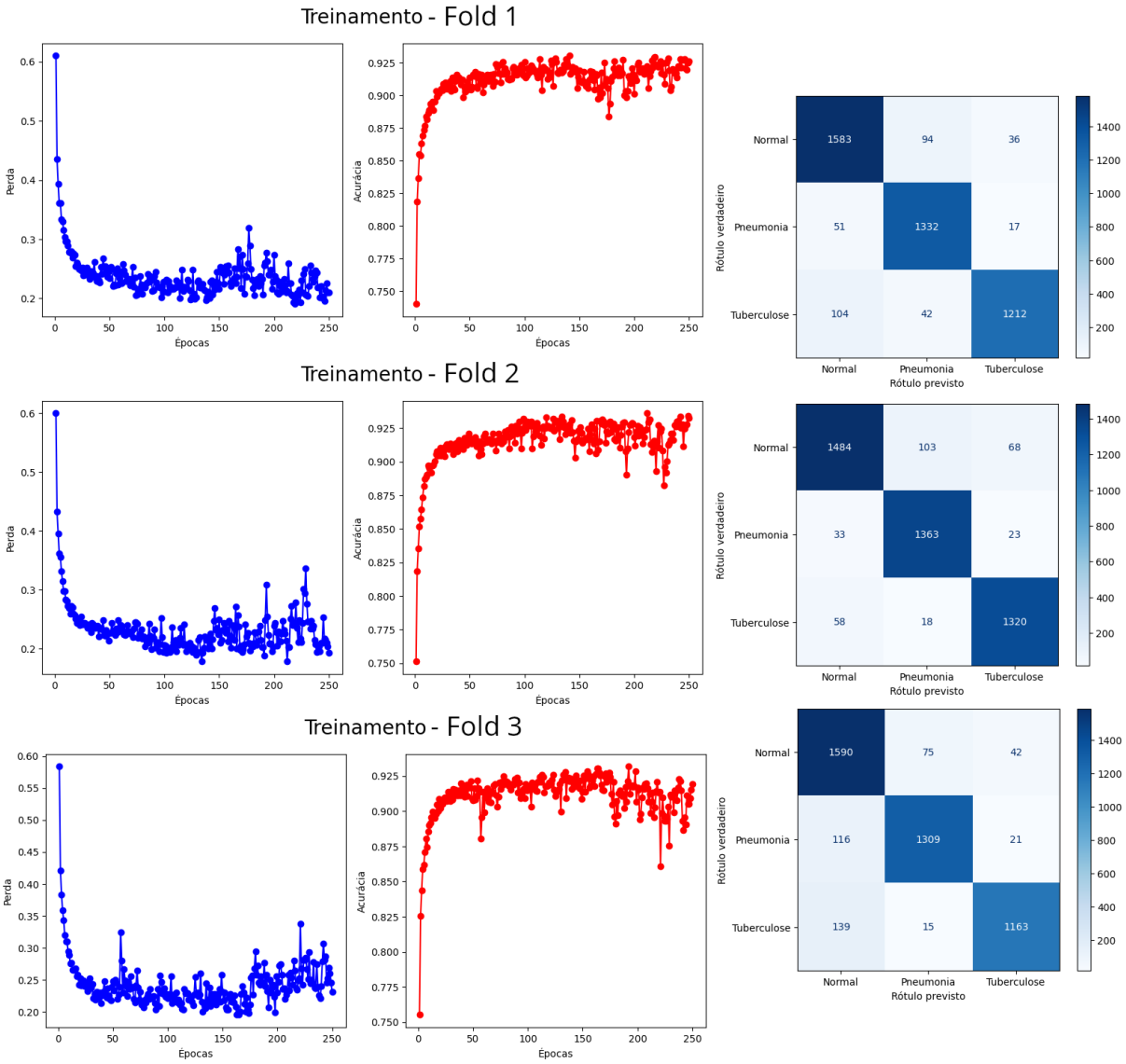
Métricas	**Média**
Acurácia	92.13%
Precisão	92.26%
Recall	92.13%
F1	92.14%

Fonte: o autor.

Visto que o modelo *Efficient* KAN resultou em 92% nas médias das métricas de avaliação, mostrado pela Tabela 5.26, mesmo sendo um bom resultado, superando a porcentagem de 90%, comparando com os resultados da aplicação do modelo CNN, a diferença foi de 3%.

Para a implementação do modelo *Fast* KAN com a técnica *KFold* definido para três *Folds*, a Figura 5.37 apresenta o comportamento da perda, mostrado pelos gráficos da esquerda, e da acurácia, mostrado pelos gráficos da direita, bem como as matrizes de confusão para cada *Fold*. Nessa aplicação, foram mantidos os parâmetros de treinamento, sem atualização dinâmica da taxa de aprendizagem, e a arquitetura da última implementação do modelo *Fast* KAN da segunda etapa. Nas Tabelas 5.27 e 5.28 são apresentados os resultados da aplicação do modelo *Fast* KAN nas imagens de teste, por meio das métricas de avaliação Acurácia, Precisão, *Recall* e pontuação *F1*.

Figura 5.36 – Treinamento e matriz de confusão para o modelo *Efficient* KAN.



Fonte: o autor.

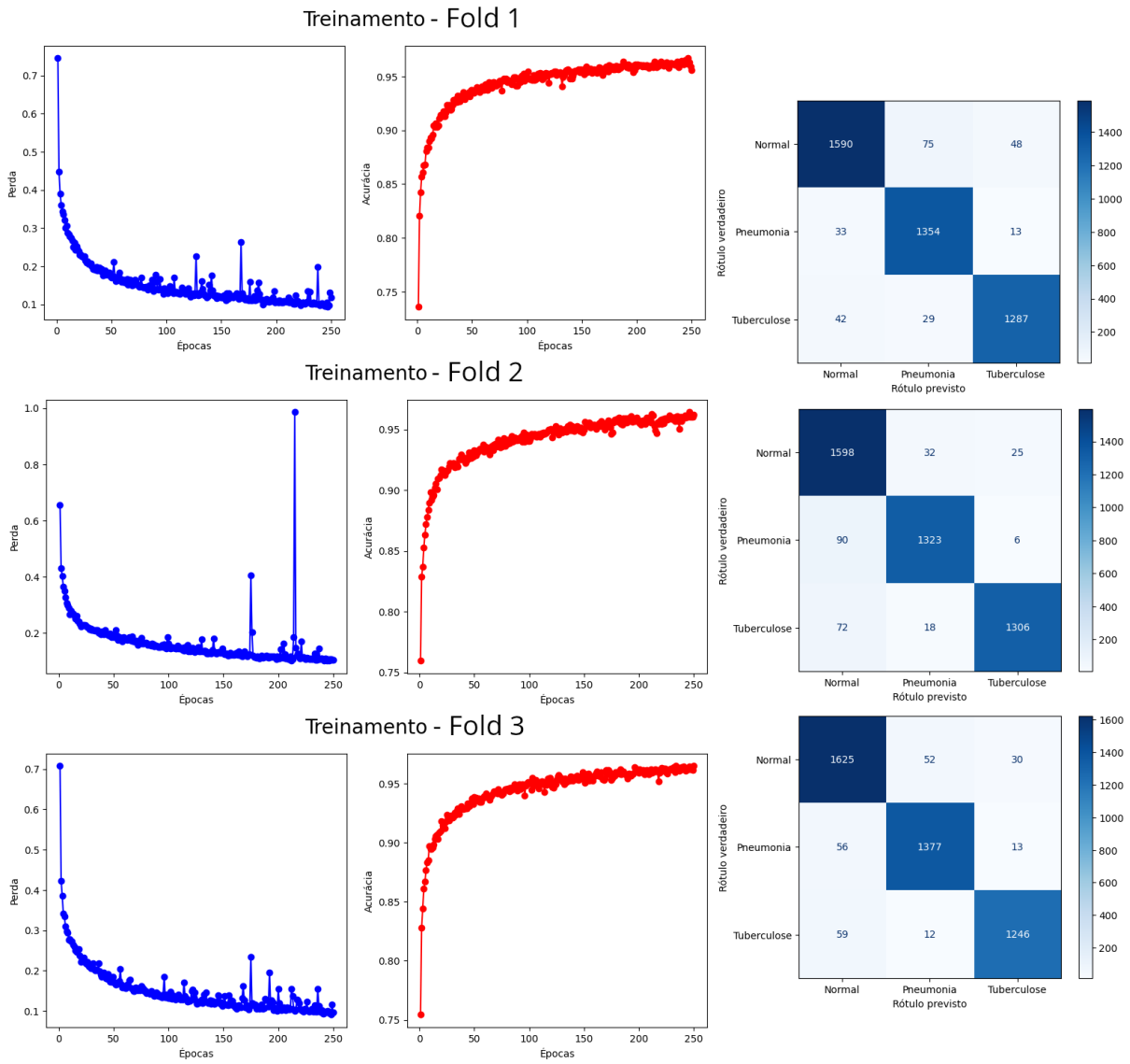
Tabela 5.27 – Métricas de avaliação para cada *Fold* no modelo *Fast* KAN.

Folds	**Acurácia**	**Precisão**	***Recall***	**F1**
1	94.63%	94.67%	94.63%	94.63%
2	95.03%	95.06%	95.03%	95.04%
3	94.56%	94.71%	94.56%	94.58%

Fonte: o autor.

Como demonstrado pelas médias das métricas de avaliação, resultados próximos de 95% foram encontrados para os modelos *Fast* KAN e CNN. Portanto, o destaque deste trabalho está no modelo *Fast* KAN, que não somente igualou os resultados com a arquitetura CNN, modelo majoritário na literatura, como também o fez com um menor tempo, com diferença de 12 horas, mostrado na Tabela 5.29.

Figura 5.37 – Treinamento e matriz de confusão para o modelo *Fast* KAN.



Fonte: o autor.

Tabela 5.28 – Média das métricas de avaliação para o modelo *Fast* KAN.

Métricas	**Média**
Acurácia	94.74%
Precisão	94.81%
Recall	94.74%
F1	94.75%

Fonte: o autor.

Tabela 5.29 – Tempo de execução para as implementações da 3^a etapa.

CNN	***Efficient* KAN**	***Fast* KAN**
48 h 40 min	41 h 17 min	36 h 08 min

Fonte: o autor.

Capítulo 6

CONCLUSÕES

Neste trabalho, modelos de aprendizado de máquina foram implementados para classificação das doenças pulmonares Pneumonia e Tuberculose por meio de imagens de radiografia do tórax, sendo eles, CNN, modelo baseado no teorema da aproximação universal e KAN (*Efficient* e *Fast*), modelo baseado no teorema de representação de Kolmogorov-Arnold.

Dentre as diversas técnicas de aquisição de imagens, radiografia do tórax, ou raio-x, foi escolhida devido a sua eficiência, simplicidade, baixo custo e no uso constante em avaliação de modelos na literatura voltados para o campo da saúde. Mesmo que existam diversas outras técnicas de aquisição de imagens utilizadas para auxiliar os profissionais da saúde no diagnóstico e tratamento de doenças respiratórias, como, por exemplo, tomografia computadorizada e ressonância magnética, é relatado no Capítulo 3 que países de renda média-baixa e renda baixa são os mais afetados por mortes ocasionadas por estas doenças, dificultando o uso de técnicas mais avançadas onde o custo elevado é a principal barreira em países com crises econômicas graves.

Técnicas de aprendizado de máquina são amplamente utilizadas para aprimorar a análise e o acompanhamento dos pacientes pelos profissionais da saúde, visto que, no contexto deste trabalho, a avaliação precisa em relação à distinção entre as doenças pulmonares a partir de imagens pode ser desafiadora para profissionais sem e com experiência. Dentre estas técnicas, destacam-se os algoritmos de aprendizado de máquina baseados em Redes Neurais Artificiais, como, por exemplo, os modelos CNN e KAN (*Efficient* e *Fast*).

A arquitetura CNN é uma das mais utilizadas na área da visão computacional, onde, as camadas convolucionais inseridas nesta estrutura possibilitam a análise de imagens para gerar novos mapas de características, que representam a saída de cada camada convolucional contendo características específicas da imagem. Baseado no Teorema da Aproximação Universal, que apresenta a possibilidade de aproximar funções contínuas em relação às operações não-linearidade, as funções de ativação são fixas nos neurônios artificiais e os pesos das conexões são lineares, onde o objetivo é minimizar a diferença entre a saída predita e a

desejada, por meio das atualizações dos pesos em relação aos cálculos das derivadas parciais, que dependerá do otimizador escolhido.

A arquitetura KAN foi desenvolvida em 2024 e descrita por seus criadores como uma alternativa promissora, onde a possibilidade de modelar funções complexas faz com que essa estrutura identifique correlações para problemas específicos, como, por exemplo, classificação de imagens. Baseado no Teorema de Representação de Kolmogorov-Arnold, que apresenta a possibilidade de representar funções contínuas em relação ao somatório de funções unidimensionais compostas, os neurônios artificiais somente realizam operações de somatório das informações recebidas dos cálculos das funções de ativação ajustáveis nos pesos das conexões. Semelhante ao modelo CNN, o objetivo é encontrar valores para os pesos das conexões que possibilitam encontrar um valor reduzido para a função de custo por meio das derivadas parciais, onde o cálculo dependerá da escolha do otimizador.

Como apresentado no Capítulo 2, a arquitetura KAN foi utilizada para desenvolver diversos outros algoritmos com finalidades distintas. Dentre estes algoritmos, foram escolhidos os modelos *Efficient* KAN, devido às adaptações realizadas na implementação para garantir melhor eficiência sem alterar as bases da arquitetura KAN, e *Fast* KAN, visto a possibilidade de uso de diferentes funções unidimensionais, neste modelo (Baseado no *Efficient* KAN) foram utilizadas as funções de base radial, em vez das funções *B-spline* do modelo anterior, para deixar as operações de treinamento mais rápido.

Para simular condições reais que podem ocorrer durante a captura das imagens, como também para diversificar os dados para evitar situações de *overfitting*, foram utilizadas as técnicas de aumento de dados Rotação (Inclinação do paciente), Deslocamento (Posição do equipamento), Escala (Diferenças no zoom), Cisalhamento (Distorções pelo ângulo de captura), Brilho (Diferenças de intensidade), Contraste (Diferenças na densidade), Saturação (Diferenças de iluminação), Matiz (Diferenças no esquema de coloração) e Ruído (Ocorrência de anomalia durante o processo de captura).

Como descrito nos Capítulos 4 e 5, foram realizadas três etapas de experimentos de treinamento-validação. Para a primeira etapa, foi selecionado um conjunto amostral contendo 1000 imagens, sendo 700 imagens para treinamento e 300 imagens para teste, sendo esta separação realizada pela técnica *Holdout*, visando testar parâmetros de treinamento e arquiteturas personalizadas para os modelos. Para a segunda etapa, foi utilizado o banco de dados completo contendo 13411 imagens, sendo 9387 imagens de treinamento e 4024 imagens para teste, separação esta realizada pela técnica *Holdout*, onde os melhores parâmetros e arquiteturas testados na primeira etapa foram executados nesta parte. Para a terceira etapa, foram mantidos o banco de dados completo, as arquiteturas e os parâmetros utilizados na segunda etapa, mas sem o uso da técnica *StepLR* para atualização dinâmica da taxa de aprendizagem. Nesta última etapa, o diferencial foi o uso da técnica de validação cruzada *KFold* para separação dos subconjuntos de treinamento e teste, sendo definido como 3 *Folds*

para cada modelo, ou seja, 3 conjuntos com diferentes subconjuntos de treinamento e teste para cada modelo.

Comparando os resultados apresentados no Capítulo 5, ignorando as duas primeiras implementações para o modelo CNN, na primeira etapa, não ocorreram grandes variações entre as porcentagens das métricas de avaliação dos modelos, onde resultados próximos de 90% foram identificados em todas as aplicações. Para a segunda etapa, o modelo CNN alcançou 97% nas métricas de avaliação, enquanto os algoritmos *Efficient* e *Fast* KAN chegou em valores próximos de 93%. Mesmo que, nesta segunda etapa, os modelos baseados na arquitetura KAN não tenham apresentado resultados melhores que o algoritmo CNN, ainda podem ser considerados bons resultados, visto que todos os modelos ultrapassaram a marca dos 90% nas métricas de avaliação. Para a terceira etapa, o destaque está com o modelo *Fast* KAN, que igualou os resultados das métricas de avaliação, próximos aos 94%, com o algoritmo CNN. Em contrapartida, o modelo *Efficient* KAN chegou em 92% nos valores das métricas de avaliação, sendo um resultado aproximado dos outros modelos, tendo ultrapassado a marca dos 90%.

Portanto, com as aplicações realizadas em diversas situações com seus respectivos resultados e com as descrições realizadas pelos autores Liu et al. (2024b), Hou e Zhang (2024) e Li (2024), é possível afirmar que os modelos, *Efficient* e *Fast* KAN, são técnicas promissoras em relação à classificação de imagens, como também possuindo um tempo de execução mais rápido, visto que, em todas as aplicações dos modelos baseados na arquitetura KAN, o tempo de execução foi melhor que aqueles apresentados pela técnica CNN, onde o algoritmo *Fast* KAN obteve o menor tempo.

Para trabalhos futuros considerando somente o campo da saúde, mais experimentos em relação à classificação de imagens devem ser realizados, mas considerando outras situações demonstradas na lista a seguir:

- Adicionar mais problemas respiratórios no banco de dados, aumentando a quantidade de classes;
- Considerar outros problemas de saúde que podem ocorrer em outras partes do corpo;
- Adicionar mais operações na estrutura KAN, como camadas convolucionais, operações de redimensionamento e operações de desativação temporária de neurônios artificiais;
- Testar diferentes otimizadores, funções de ativação e funções de custo;
- Testar diferentes funções unidimensionais, respeitando o Teorema de Representação de Kolmogorov-Arnold.

REFERÊNCIAS

AGHAEI, A. A. fkan: Fractional kolmogorov-arnold networks with trainable jacobi basis functions. *Neurocomputing*, Elsevier, p. 129414, 2025. Citado na página 27.

ALSHMRANI, G. M. M. *et al.* A deep learning architecture for multi-class lung diseases classification using chest x-ray (cxr) images. *Alexandria Engineering Journal*, Elsevier, v. 64, 2023. Citado na página 26.

ANTHIMOPOULOS, M. *et al.* Lung pattern classification for interstitial lung diseases using a deep convolutional neural network. *IEEE transactions on medical imaging*, IEEE, v. 35, n. 5, 2016. Citado na página 21.

ARIAS-LONDOÑO, J. D. *et al.* Artificial intelligence applied to chest x-ray images for the automatic detection of covid-19. a thoughtful evaluation approach. *Ieee Access*, IEEE, v. 8, 2020. Citado na página 25.

ARSOMNGERN, P. *et al.* Computer-aided diagnosis for lung lesion in companion animals from x-ray images using deep learning techniques. In: IEEE. *2019 IEEE 10th International Conference on Awareness Science and Technology (iCAST)*. [S.l.], 2019. Citado na página 46.

AUSAWALAITHONG, W. *et al.* Automatic lung cancer prediction from chest x-ray images using the deep learning approach. In: IEEE. *2018 11th biomedical engineering international conference (BMEiCON)*. [S.l.], 2018. Citado na página 25.

BALTRUSCHAT, I. M. *et al.* When does bone suppression and lung field segmentation improve chest x-ray disease classification? In: IEEE. *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)*. [S.l.], 2019. Citado na página 46.

BISHOP, C. M.; NASRABADI, N. M. *Pattern recognition and machine learning*. [S.l.]: Springer, 2006. v. 4. Citado 2 vezes nas páginas 39 e 40.

BLEALTAN. *Efficient-KAN*. 2024. <https://github.com/Blealtan/efficient-kan>. Repositório GitHub, acessado em 05/09/2024. Citado 2 vezes nas páginas 46 e 54.

BODNER, A. D. *et al.* Convolutional kolmogorov-arnold networks. *arXiv preprint arXiv:2406.13155*, 2024. Citado na página 27.

BOZORGASL, Z.; CHEN, H. Wav-kan: Wavelet kolmogorov-arnold networks. *arXiv preprint arXiv:2405.12832*, 2024. Citado na página 27.

BRESSON, R. *et al.* Kagnns: Kolmogorov-arnold networks meet graph learning. *arXiv preprint arXiv:2406.18380*, 2024. Citado na página 27.

CARLO, G. D.; MASTROPIETRO, A.; ANAGNOSTOPOULOS, A. Kolmogorov-arnold graph neural networks. *arXiv preprint arXiv:2406.18354*, 2024. Citado na página 27.

CHAUHAN, A.; CHAUHAN, D.; ROUT, C. Role of gist and phog features in computer-aided diagnosis of tuberculosis without segmentation. *PloS one*, Public Library of Science San Francisco, USA, v. 9, n. 11, p. e112980, 2014. Citado na página 53.

CHEON, M. Demonstrating the efficacy of kolmogorov-arnold networks in vision tasks. *arXiv preprint arXiv:2406.14916*, 2024. Citado na página 27.

CHOKCHAITHANAKUL, W.; PUNYABUKKANA, P.; CHUANGSUWANICH, E. Adaptive image preprocessing and augmentation for tuberculosis screening on out-of-domain chest x-ray dataset. *IEEE Access*, IEEE, v. 10, 2022. Citado na página 46.

COHEN, S. B. *et al.* Alveolar macrophages provide an early mycobacterium tuberculosis niche and initiate dissemination. *Cell host & microbe*, Elsevier, v. 24, n. 3, 2018. Citado na página 33.

DIAGNÓSTICOS, S. C. *Ultrassonografia Obstétrica*. 2024. Disponível em: <https://santaclaradiagnosticos.com.br/ultrassonografia-obstetrica-2>. Acesso em: 14/09/2024. Citado na página 35.

DROKIN, I. Kolmogorov-arnold convolutions: Design principles and empirical studies. *arXiv preprint arXiv:2407.01092*, 2024. Citado na página 27.

DUMOULIN, V.; VISIN, F. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016. Citado 2 vezes nas páginas 43 e 44.

EINSTEIN, H. I. A. *Pneumonia Viral: Quando o Vírus da Gripe Agridem os Pulmões*. 2024. Disponível em: <https://vidasaudavel.einstein.br/pneumonia-viral-quando-o-virus-da-gripe-agride-os-pulmoes>. Acesso em: 10/09/2024. Citado na página 32.

ELAZIZ, M. A.; FARES, I. A.; ASEERI, A. O. Ckan: Convolutional kolmogorov–arnold networks model for intrusion detection in iot environment. *IEEE Access*, IEEE, 2024. Citado na página 27.

EUSEBI, P. Diagnostic accuracy measures. *Cerebrovascular Diseases*, S. Karger AG, v. 36, n. 4, p. 267–272, 2013. Citado na página 47.

FAWCETT, T. An introduction to roc analysis. *Pattern recognition letters*, Elsevier, v. 27, n. 8, 2006. Citado 2 vezes nas páginas 47 e 48.

FERNÁNDEZ-MIRANDA, P. M. *et al.* A retrospective study of deep learning generalization across two centers and multiple models of x-ray devices using covid-19 chest-x rays. *Scientific Reports*, Nature Publishing Group UK London, v. 14, n. 1, 2024. Citado na página 22.

FIRSOV, N. *et al.* Hyperkan: Kolmogorov–arnold networks make hyperspectral image classifiers smarter. *Sensors (Basel, Switzerland)*, v. 24, n. 23, p. 7683, 2024. Citado na página 27.

FONSECA, E. K. U. N. *et al.* Tomografia computadorizada de tórax no diagnóstico de covid-19 em pacientes com resultado falso-negativo na rt-pcr. *einstein (São Paulo)*, SciELO Brasil, v. 19, 2021. Citado na página 35.

FOUNDATION, P. S. *time – Time access and conversions*. 2024. Disponível em: <https://docs.python.org/3/library/time.html>. Acesso em: 05/09/2024. Citado na página 54.

GENET, R.; INZIRILLO, H. Tkan: Temporal kolmogorov-arnold networks. *arXiv preprint arXiv:2405.07344*, 2024. Citado na página 27.

GÉRON, A. *Mãos à Obra: Aprendizado de Máquina com Scikit-Learn & TensorFlow*. [S.l.]: Alta Books, 2019. Citado na página 42.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. Citado 4 vezes nas páginas 39, 40, 41 e 43.

GORDON, W. J.; RIESENFELD, R. F. B-spline curves and surfaces. In: *Computer aided geometric design*. [S.l.]: Elsevier, 1974. p. 95–126. Citado na página 44.

HAMAL, S. *et al.* A comparative analysis of machine learning algorithms for detecting covid-19 using lung x-ray images. *Decision Analytics Journal*, Elsevier, v. 11, 2024. Citado na página 25.

HAQUE, A. ul *et al.* Pneumonia classification: A limited data approach for global understanding. *Heliyon*, Elsevier, v. 10, n. 4, 2024. Citado na página 21.

HARRIS, C. R. *et al.* Array programming with NumPy. *Nature*, Springer Science and Business Media LLC, v. 585, n. 7825, 2020. Citado na página 54.

HASAN, M. Z. *et al.* Fast and efficient lung abnormality identification with explainable ai: A comprehensive framework for chest ct scan and x-ray images. *IEEE Access*, IEEE, v. 12, 2024. Citado na página 22.

HAYKIN, S. *Neural networks and learning machines, 3/E*. [S.l.]: Pearson Education India, 2009. Citado 2 vezes nas páginas 36 e 49.

HE, H.; GARCIA, E. A. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, Ieee, v. 21, n. 9, p. 1263–1284, 2009. Citado na página 48.

HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. *Neural networks*, Elsevier, v. 2, n. 5, 1989. Citado na página 44.

HOU, Y.; ZHANG, D. A comprehensive survey on kolmogorov arnold networks (kan). *arXiv preprint arXiv:2407.11075*, 2024. Citado 2 vezes nas páginas 22 e 99.

HUNTER, J. D. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, IEEE COMPUTER SOC, v. 9, n. 3, 2007. Citado na página 54.

IGALI, A.; SHAMOI, P. Image classification using fuzzy pooling in convolutional kolmogorov-arnold networks. In: IEEE. *2024 Joint 13th International Conference on Soft Computing and Intelligent Systems and 25th International Symposium on Advanced Intelligent Systems (SCIS&ISIS)*. [S.l.], 2024. p. 1–6. Citado na página 27.

IOFFE, S. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. Citado na página 43.

JIAO, L.; ZHAO, J. A survey on the new generation of deep learning in image processing. *Ieee Access*, IEEE, v. 7, 2019. Citado na página 22.

KERMANY, D. S. *et al.* Identifying medical diagnoses and treatable diseases by image-based deep learning. *cell*, Elsevier, v. 172, n. 5, 2018. Citado 2 vezes nas páginas 34 e 52.

KIAMARI, M.; KIAMARI, M.; KRISHNAMACHARI, B. Gkan: Graph kolmogorov-arnold networks. *arXiv preprint arXiv:2406.06470*, 2024. Citado na página 27.

KIEU, P. N. *et al.* Applying multi-cnns model for detecting abnormal problem on chest x-ray images. In: IEEE. *2018 10th International Conference on Knowledge and Systems Engineering (KSE)*. [S.l.], 2018. Citado na página 22.

KIM, C.-M.; HONG, E. J.; PARK, R. C. Chest x-ray outlier detection model using dimension reduction and edge detection. *IEEE Access*, IEEE, v. 9, 2021. Citado na página 26.

KINGMA, D. P. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. Citado 2 vezes nas páginas 41 e 42.

KIRAN, S.; JABEEN, D. I. *Dataset of Tuberculosis Chest X-rays Images*. Mendeley Data, 2024. Disponível em: <https://data.mendeley.com/datasets/8j2g3csprk/2>. Citado 2 vezes nas páginas 53 e 54.

KOHAVI, R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: MONTREAL, CANADA. *Ijcai*. [S.l.], 1995. v. 14, n. 2, p. 1137–1145. Citado na página 49.

KOLMOGOROV, A. N. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. In: RUSSIAN ACADEMY OF SCIENCES. *Doklady Akademii Nauk*. [S.l.], 1957. v. 114, n. 5. Citado na página 44.

LAI, R. *et al.* Heterogeneity in lung macrophage control of mycobacterium tuberculosis is modulated by t cells. *Nature Communications*, Nature Publishing Group UK London, v. 15, n. 1, 2024. Citado na página 21.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *nature*, Nature Publishing Group UK London, v. 521, n. 7553, 2015. Citado 2 vezes nas páginas 22 e 41.

LECUN, Y. *et al.* Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, Ieee, v. 86, n. 11, 1998. Citado 2 vezes nas páginas 22 e 42.

LI, Z. Kolmogorov-arnold networks are radial basis function networks. *arXiv preprint arXiv:2405.06721*, 2024. Citado 5 vezes nas páginas 27, 46, 54, 65 e 99.

LIU, Y. *et al.* Rethinking computer-aided tuberculosis diagnosis. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2020. p. 2646–2655. Citado na página 53.

LIU, Z. *et al.* Kan 2.0: Kolmogorov-arnold networks meet science. *arXiv preprint arXiv:2408.10205*, 2024. Citado na página 27.

____. Kan: Kolmogorov-arnold networks. *arXiv preprint arXiv:2404.19756*, 2024. Citado 6 vezes nas páginas 22, 26, 44, 45, 54 e 99.

MALIK, H. *et al.* A novel fusion model of hand-crafted features with deep convolutional neural networks for classification of several chest diseases using x-ray images. *IEEE Access*, IEEE, v. 11, 2023. Citado na página 26.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, Springer, v. 5, p. 115–133, 1943. Citado 3 vezes nas páginas 36, 37 e 38.

MEHLIG, B. *Machine learning with neural networks: an introduction for scientists and engineers*. [S.l.]: Cambridge University Press, 2021. Citado na página 43.

MOORE, K. L.; DALLEY, A. F.; AGUR, A. M. *Anatomia Orientada para a Clínica, 7/E*. [S.l.]: Guanabara Koogan, 2014. Citado 2 vezes nas páginas 34 e 35.

MUÑOZ-ASEGUINOLAZA, U. *et al.* Convolutional neural network-based classification and monitoring models for lung cancer detection: 3d perspective approach. *Heliyon*, Elsevier, v. 9, n. 11, 2023. Citado na página 25.

MURPHY, K. P. *Machine learning: a probabilistic perspective*. [S.l.]: MIT press, 2012. Citado na página 40.

NAIR, V.; HINTON, G. E. Rectified linear units improve restricted boltzmann machines. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. [S.l.: s.n.], 2010. Citado na página 39.

PASZKE, A. *et al.* Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, v. 32, 2019. Citado na página 54.

PATEL, A. N. *et al.* An explainable transfer learning framework for multi-classification of lung diseases in chest x-rays. *Alexandria Engineering Journal*, Elsevier, v. 98, 2024. Citado na página 21.

PATWA, A.; SHAH, A. Anatomy and physiology of respiratory system relevant to anaesthesia. *Indian journal of anaesthesia*, Medknow, v. 59, n. 9, 2015. Citado 2 vezes nas páginas 29 e 30.

PEDREGOSA, F. *et al.* Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, v. 12, 2011. Citado na página 54.

PRATT, L. Y. Discriminability-based transfer between neural networks. *Advances in neural information processing systems*, v. 5, 1992. Citado na página 25.

RAHMAN, T. *et al.* Reliable tuberculosis detection using chest x-ray with deep learning, segmentation and visualization. *Ieee Access*, IEEE, v. 8, 2020. Citado 3 vezes nas páginas 22, 52 e 53.

RASCHKA, S.; MIRJALILI, V. *Python machine learning: Machine learning and deep learning with Python, scikit-learn, and TensorFlow, 2/E*. [S.l.]: Packt Publishing Ltd, 2017. Citado na página 40.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, American Psychological Association, v. 65, n. 6, p. 386, 1958. Citado 2 vezes nas páginas 37 e 38.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *nature*, Nature Publishing Group UK London, v. 323, n. 6088, 1986. Citado na página 38.

SAITO, T.; REHMSMEIER, M. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PloS one*, Public Library of Science San Francisco, CA USA, v. 10, n. 3, p. e0118432, 2015. Citado na página 48.

SANTOS, V. S. dos. *Sistema respiratório*. 2024. <https://brasilescola.uol.com.br/biologia/sistema-respiratorio.htm>. Brasil Escola, acessado em 06/09/2024. Citado 2 vezes nas páginas 29 e 30.

SATHITRATANACHEEWIN, S.; SUNANTA, P.; PONGPIRUL, K. Deep learning for automated classification of tuberculosis-related chest x-ray: dataset distribution shift limits diagnostic performance generalizability. *Heliyon*, Elsevier, v. 6, n. 8, 2020. Citado na página 22.

SAÚDE, M. da. *Pneumonia*. 2024. Disponível em: <https://bvsms.saude.gov.br/pneumonia-5>. Acesso em: 10/09/2024. Citado na página 32.

SAÚDE, O. M. da. *The Top 10 Causes of Death*. 2023. Disponível em: <https://www.who.int/news-room/fact-sheets/detail/the-top-10-causes-of-death>. Acesso em: 19/08/2024. Citado 2 vezes nas páginas 21 e 31.

____. *Pneumonia*. 2024. Disponível em: <https://www.who.int/health-topics/pneumonia#tab=tab_1>. Acesso em: 10/09/2024. Citado na página 32.

____. *Tuberculosis*. 2024. Disponível em: <https://www.who.int/health-topics/tuberculosis#tab=tab_1>. Acesso em: 12/09/2024. Citado 2 vezes nas páginas 33 e 34.

SOMVANSHI, S. *et al.* A survey on kolmogorov-arnold network. *arXiv preprint arXiv:2411.06078*, 2024. Citado na página 27.

VACA-RUBIO, C. J. *et al.* Kolmogorov-arnold networks (kans) for time series analysis. *arXiv preprint arXiv:2405.08790*, 2024. Citado na página 27.

VIEIRA, P. de A. *Detecção de doenças pulmonares utilizando aprendizado profundo em imagens de raios-x*. Dissertação (Mestrado) — Universidade Federal do Piauí, 2021. Citado na página 32.

XIE, X. *et al.* Kansformer for scalable beamforming. *arXiv preprint arXiv:2410.20690*, 2024. Citado na página 27.

XU, J. *et al.* Fourierkan-gcf: Fourier kolmogorov-arnold network–an effective and efficient feature transformation for graph collaborative filtering. *arXiv preprint arXiv:2406.01034*, 2024. Citado na página 27.

XU, K.; CHEN, L.; WANG, S. Kolmogorov-arnold networks for time series: Bridging predictive power and interpretability. *arXiv preprint arXiv:2406.02496*, 2024. Citado na página 27.

ZHOU; CHELLAPPA. Computation of optical flow using a neural network. In: IEEE. *IEEE 1988 international conference on neural networks*. [S.l.], 1988. p. 71–78. Citado na página 43.

ZHOU, C. *et al.* Covid-19 detection based on image regrouping and resnet-svm using chest x-ray images. *Ieee Access*, IEEE, v. 9, 2021. Citado na página 26.

APÊNDICE A

Links pertinentes

O link a seguir refere-se ao local onde pode ser encontrado o repositório com as implementações dos modelos, bem como um link do banco de dados armazenado na plataforma *Kaggle*: https://github.com/AlxDev-hub/Classificacao-KAN-DoencasPulmonares.